

# Tutorial 4:

## Adversarial Search

### Question 1

Problem representation of Tic-tac-toe game:

What exists in this world?

Players (O= Naught, X= Cross), B=Blanks, a 3x3 board.

OR 0=Blank, 1=Cross, -1=Naught

How to represent states?

A state = a 3x3 array Board[i][j] where  $0 \leq i, j \leq 2$

### Question 2

Terminal test, there exist i,j such that  $0 \leq n \leq 2$ ,  $0 \leq m \leq 2$ ,

1. Board[i][n] = X or Board[i][n] = O or
2. Board[n][j] = X or Board[n][j] = O or
3. Board[n][n] = X or Board[n][n] = O or
4. Board[n][2-n] = X or Board[n][2-n] = O or
5. Board[n][m] = X or O

### Question 3

A successor function is a \*problem-dependent\* function that takes a state and returns all of the possible successors of that state. That is, given a state, a successor function produces all of the legal 'moves' according to the problem transition functions.

function Successors(state,p=Player) returns (action, state) pairs

T = []

for i,j where  $0 \leq i, j \leq 2$

if state[i][j] = 0 then

Board = Copy(state)

Board[i][j] = p

addToList(T,(action<sub>ijp</sub>,Board))

return T

## Question 4

Heuristic Function (also called evaluation function):

Define various features of the state, give weights to the features, and calculate weighted average of the features (e.g. in chess this could be numbers of each kind of piece on the board with weights for the different types)

Features:

f1: in a row, column, or diagonal, one X and empty cells: weight 0.1

f2: in a row, column, or diagonal, two X and an empty cell: weight 0.2

Function definition

$f1(s)$  = the number of feature f1 in state s

$f2(s)$  = the number of feature f2 in state s

$Heuristic(s) = f1(s) * 0.1 + f2(s) * 0.5$

## Question 5

Two-ply search means that your resulting tree will contain three levels \*including\* the root node.

Each node will have an expected utility or heuristic function

The minimax algorithm does not assign nodes utility values on the first pass through a tree. The utility value of any non-leaf node is a property calculated from the utility values of its children.

The algorithm description says to assign '-' to maximising nodes and '+' to minimising nodes; what is meant here is to assign '-infinity' and '+infinity' so that the first max or min performed on that node is guaranteed to replace the node's value. See also the pseudocode for Minimax in the textbook and lecture notes.

Methods:

Define various features of the state

Give weights to the features

Calculate weighted average of the features:  $Eval(s) = \sum_{i,j} (w_{ij} f_{ij}(s))$

E.g., in chess,  $f_j$  could be the numbers of each kind of piece on the board and  $w$  could be the weights of the pieces (1 for pawn, 3 for bishop, etc).

MiniMax search algorithm from lecture4 (modify this to answer question 3):

Function MAXSearch(state) returns action

$v = MAX-VALUE(state)$

return the action in SUCCESSORS(state) with v

Function MAX-VALUE(state) return utility

if TERMINAL(state) return UTILITY(state)

$v = -infinity$

for a, s in SUCCESSORS(state)

$v = MAX(v, MIN-VALUE(s))$

return v

```

Function MIN-VALUE(state) return utility
  if TERMINAL(state) return UTILITY(state)
  v = + infinity
  for a, s in SUCCESSORS(state)
    v=MIN(v, MAX-VALUE(s))
  return v

```

In the following, underlined statements are for  $\alpha\beta$ -pruning.

Decision function: an action to perform in node A

function TwoPly-MiniMax-Search(state) returns an action

```

  inputs: state, current state in game
  v  $\leftarrow$  Max-Value(state, -infinite, +infinite)
  return the action in Successors(state, X) with value v.

```

function Max-Value(state,  $\alpha$ ,  $\beta$ ) returns a utility value

```

  inputs: state, current state in game
   $\alpha$ , the value of the best alternative for Max along the path to state
   $\beta$ , the value of the best alternative for Min along the path to state
  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$  -infinite
  for (action, sstate) in Successors(state, X) do
    v  $\leftarrow$  Max(v, Min-Value(sstate,  $\alpha$ ,  $\beta$ ))
    if v  $\geq$   $\beta$  then return v
     $\alpha \leftarrow$  Max( $\alpha$ , v)
  return v

```

function Min-Value(state,  $\alpha$ ,  $\beta$ ) returns a utility value

```

  inputs: state, current state in game
   $\alpha$ , the value of the best alternative for Max along the path to state
   $\beta$ , the value of the best alternative for Min along the path to state
  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$  -infinite
  for (action, sstate) in Successors(state, X) do
    v  $\leftarrow$  Min(v, Heuristics(sstate))
    if v  $\geq$   $\beta$  then return v
     $\alpha \leftarrow$  Max( $\alpha$ , v)
  return v

```

The heuristics line here is the one that indicates the cutoff in the two-ply search

## Question 6

Alpha-beta pruning for player MAX:

MAX is looking for max values.

- Always choose the node with the maximum value:  $\text{value of a node} = \text{MaxValue}(\text{Successors}(\text{node}))$
- i.e., no need to further evaluate successor nodes with value smaller than current max value.

MIN is looking for min values.

- Always choose the node with the smallest value:  $\text{value of a node} = \text{MinValue}(\text{Successors}(\text{node}))$ .
- i.e., no need to further evaluate successor nodes with value larger than current min value.