Informatik II (D-ITET)
Tutorial 10

TA: Hông-Ân Cao, E-mail: hong-an.cao@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- **Exercise 9: Solution discussion**

- Exercise 10: Overview (Mergesort, Hanoi Towers, Reversi)
Solution Ex9.Q1a,b,c – Game Tree Min-Max

MAX strategy

Height: 4
Search Depth: 3
Best path: left

MAX
MIN
MAX
MIN

Search Depth: 3
Best path: left
Solution Ex9.Q1d – The α-β algorithm

- **The α-β algorithm**
  - Reduces the game tree through pruning, but delivers the Min-Max value of the root in the same way as the Min-Max algorithm
  - The MinMax algorithm evaluates the whole search tree. In this case, nodes that don't influence the outcome (choice of the branch at the root) are also evaluated. The Alpha-Beta search ignores those nodes.

- **α**
  - The smallest known value of all MAX evaluations of the MIN children
  - Is relevant for the evaluation of MAX nodes
    - Evaluation of the successors can be aborted as soon as the computed return value is below α

- **β**
  - The largest known value of all MIN evaluations of the MAX children
  - Is relevant for the evaluation of MIN nodes
    - Evaluation of the successors can be aborted as soon as the computed return value is above β
L9.A2d – The Tree

MAX

MIN

MAX

MIN
function minimax(node, depth, maximizingPlayer)
  if depth = 0 or node is a terminal node
    return the heuristic value of node
  if maximizingPlayer
    bestValue := −∞
    for each child of node
      v := minimax(child, depth − 1, FALSE)
      bestValue := max(bestValue, v)
    return bestValue
  else (* minimizing player *)
    bestValue := +∞
    for each child of node
      v := minimax(child, depth − 1, TRUE)
      bestValue := min(bestValue, v)
    return bestValue

Source: https://en.wikipedia.org/wiki/Minimax
Solution Ex9.Q2a – Min-Max

- Two helping methods:
  - \( \text{max}(...) \)
  - \( \text{min}(...) \)

- Idea: \( \text{max}() \) and \( \text{min}() \) call one another in turns

- Until we reach depth \( d \)
  - \( \text{nextMove}() \)

```java
public Coordinates nextMove(GameBoard gb){
    BestMove bestMove = null;
    bestMove = max(1, gb, 0);
    return bestMove.coord;
}

class BestMove {
    public Coordinates coord;
    public int value;
    public boolean cut;
    //whether it was cut at the
    //maximum recursion depth
    public BestMove(int value, Coordinates coord, boolean cut){
        this.value = value;
        this.coord = coord;
        this.cut = cut;
    }
}
```
private BestMove max(int maxDepth, GameBoard gb, int depth) {
    if (depth == maxDepth) { return new BestMove(eval(gb), null, true); }
    ArrayList<Coordinates> availableMoves = getMovesFor(myColor, gb); //The way we did the first week
    if (availableMoves.isEmpty()) {
        if (gb.isMoveAvailable(otherColor)) {
            BestMove result = min(maxDepth, gb, depth + 1);
            return new BestMove(result.value, null, false);
        } else { return new BestMove(finalResult(gb), null, false); } //finalResult yields the final score
    }
    BestMove bestMove = new BestMove(Integer.MIN_VALUE, null, false);
    for (Coordinates coord : availableMoves) {
        GameBoard hypothetical = gb.clone();
        hypothetical.checkMove(myColor, coord);
        hypothetical.makeMove(myColor, coord);
        BestMove result = min(maxDepth, hypothetical, depth + 1);
        if (result.value > bestMove.value) {
            bestMove.coord = coord;
            bestMove.value = result.value;
        }
        bestMove.cut = bestMove.cut || result.cut;
    }
    return bestMove;
}
Solution Ex9.Q2b – timeLimit

- Timeout per move:
  - `nextMove()` has to return a valid move before the time-out of `timeLimit` milliseconds

```java
public Coordinates nextMove(GameBoard gb) {
    long timeout = System.currentTimeMillis() + timeLimit - 10;
    BestMove bestMove = null;
    try {
        bestMove = max(1, timeout, gb, 0);
    } catch (Timeout e) {
        throw new AssertionError("Not enough time for depth 1!");
        return null;
    }
    try {
        for (int i = 2; bestMove.cut; i++) {
            bestMove = max(i, timeout, gb, 0);
        }
    } catch (Timeout e) {} 
    return bestMove.coord;
}
```
class Timeout extends Throwable{
}

private BestMove max(int maxDepth, long timeout, GameBoard gb, int depth) throws Timeout {
    if (System.currentTimeMillis() > timeout) {
        throw new Timeout();
    }

    if (depth == maxDepth) {
        return new BestMove( eval(gb), null, true );
    }

    ...

    return bestMove;
}
Solution Ex9.Q3c – Evaluation function

- Propositions for possible, static evaluations:
  - Agility
    - How many moves are possible for me / my opponent?
  - Rows
    - How many rows of connected stones are there?
    - How long are they? Their location is also interesting!
    - A fully occupied border is really good, while a long sequence in the opponent's can potentially allow for good moves
  - How many stones...
    - will be flipped by a given move and in how many directions? Are the stones lying inside the board or at the borders?
  - How many stones…
    - off a specific color are lying on the board? (That might be the evaluation function for the final game, when a thorough analysis of the search tree is possible. In the middle of the game, this might be inappropriate.)
  - Positions
    - To be evaluated on the field (e.g. corner points)
Outlook

- Exercise 9: Solution discussion
- Exercise 10: Overview (Mergesort, Hanoï Towers, Reversi)
Hints Ex10.Q1 – Mergesort

- Mergesort
  - Is a recursive and stable sort algorithm, which is based on the divide and conquer principle
  - Is was developed in 1945 by John von Neumann

- Divide and conquer principle
  - Separate the enemies to vanquish them
  - Political and military strategy
  - Was already applied in the Roman empire

John von Neumann
1903 Budapest – 1957 Washington
Hints Ex10.Q1a – Manual work

- Mergesort
  - Consider the data to be sorted as a list and decompose it into smaller lists, which will have to be sorted
  - The smaller sorted lists are merged together in a zipper pull manner, until one common list is achieved

6 5 3 1 8 7 2 4

Source: https://en.wikipedia.org/wiki/Merge_sort
Hints Ex10.Q1b – Implementation

- **ISort** defines an interface
  - **ISort.sort** takes an **ArrayList** and return a new sorted **ArrayList**

- **MergeSort.java (build)**
  - Implement the **ISort** interface
  - Tip: recursive helper method
  - Tip: one does not always build a new list, but one can play with begin-end indices
Hints Ex10.Q1c,d – Measure.java

- 10 "measured points"
  - Make sure that the random arrays are built beyond the time measurements!

- Repeat the measurements
  - Ignore both min and max (extreme values)
  - Take the average of n measurements (overall n+2 measurement runs)

- Build a diagram
  - Your favorite tool (e.g.: GNUplot, Excel, Matlab, …)
  - Deliver a graphic
  - Interpretation must add up!
Hints Ex10.Q2 – Tower of Hanoï

- In the lecture
  - Recursive solution to the problem

- The only possibility is to move the bottom-most (largest) disc from tower 1 to tower 3:
  - (a) There is nothing else on tower 1
  - (b) Tower 3 is empty

- From (a) and (b) derive:
  - All other discs are on tower 2!
  - At first, the n-1 other discs must be moved from tower 1 to tower 2

Hints Ex10.Q2 – Tower of Hanoï

- Solution for the 3-disc case
  - Name the 3 towers from left to right 1, 2, 3 and the discs from the smallest to the largest A, B, C
  - Then use the number-letter pair to indicate where a disc has to be moved
  - C2 means for example that the largest disc has to be moved to the tower in the middle.

- Steps for the solution:
  - A3, B2, A2, C3, A1, B3, A3 (7 steps)
Hints Ex10.Q2.a/b

- Identify regularities:
  - For each step in the execution of the recursive algorithm of the lecture, exactly one tower is not necessary.
  - When shifting a tower of height 4 in 15 steps, give the sequence of tower number that is not used.
Hints Ex10.Q2b,c – Pseudo-code

- Describe all "developed" algorithms in pseudo-code
  - For the starting tower of height 4
  - Are adaptations necessary when starting with a tower of height 5?
## Hints Ex10.Q3 – Reversi (Part 4)

<table>
<thead>
<tr>
<th>Player</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HumanPlayer</td>
<td>nextMove()</td>
<td>Waits for entry from command line</td>
</tr>
<tr>
<td>RandomPlayer</td>
<td>nextMove()</td>
<td>Chooses a random (but valid!) next move</td>
</tr>
<tr>
<td>GreedyPlayer</td>
<td>nextMove()</td>
<td>Chooses the next move by means of an easy and non-recursive evaluation function</td>
</tr>
<tr>
<td>MinMaxPlayer</td>
<td>nextMove()</td>
<td>Chooses the next move by means of a Minimax analysis through a new evaluation function</td>
</tr>
<tr>
<td>α-β-Player</td>
<td>nextMove()</td>
<td>Chooses the next move by means of the α-β analysis with your own evaluation function</td>
</tr>
</tbody>
</table>

- **Download**
- **Exercise 7**
- **Exercise 8**
- **Exercise 9**
- **Exercise 10**
Hints Ex10.Q3a – Reversi (Part 4)

- Build an evaluation function, which follows the α-β process, which produces the same result as the pure Min-Max method of the previous exercise sheet

- α-β algorithm
  - Beware that in this exercise the algorithm of the lecture is requested, not one of its adaptation
  - Implement a strategy for aborting a move through a throwable timeout
Reversi Tournament

- Wednesday, June 1, 2016 12:30 @CABinett (Stuz2)
- Submission:
  - Deadline: Wednesday, May 25, 2016, 23:59 (Zürich Time)
  - Submit through the Reversi Platform
  - In case of problem send an e-mail to Leyna Sadamori
  - You can work alone or in groups of two.
Have fun!

**INEFFECTIVE SORTS**

```python
# DEFINE HALFPARTITIONEDMERGESORT(LIST):
#   IF LENGTH(LIST) < 2:
#     RETURN LIST
#   PIVOT = INT(LENGTH(LIST) / 2)
#   A = HALFPARTITIONEDMERGESORT(LIST[:PIVOT])
#   B = HALFPARTITIONEDMERGESORT(LIST[PIVOT:])
#   RETURN [A, B] // HERE... SORRY.

# DEFINE FASTBEGINSORT(LIST):
#   // AN OPTIMIZED BEGINSORT
#   // RIPS IN O(N LOG N)
#   FOR N FROM 1 TO LOG(LENGTH(LIST)):
#     SHUFFLE(LIST):
#     IF ISORTED(LIST):
#       RETURN LIST
#     RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"

# DEFINE JOSHUALINTHELQUICKSORT(LIST):
#   OK. SO YOU CHOOSE A PIVOT
#   THEN DIVIDE THE LIST IN HALF
#   FOR EACH HALF:
#     CHECK TO SEE IF IT'S SORTED
#     NO, WAIT IT DOESN'T MATTER
#     COMPARE EACH ELEMENT TO THE PIVOT
#     THE BIGGER ONES GO IN A NEW LIST
#     THE EQUAL ONES GO INTO, UM
#     THE SECOND LIST FROM BEFORE
#     HANG ON, LET ME NAME THE LISTS
#     THIS IS LIST A
#     THE NEW ONE IS LIST B
#     PUT THE BIG ONES INTO LIST B
#     NOW TAKE THE SECOND LIST
#     CALL IT LIST UM, A2
#     WHICH ONE WAS THE PIVOT IN?
#     SCRATCH ALL THAT
#     IT JUST RECURSIVELY CALLS ITSELF
#     UNTIL BOTH LISTS ARE EMPTY
#     RIGHT?
#     NOT EMPTY, BUT YOU KNOW WHAT I MEAN
#     AM I ALLOWED TO USE THE STANDARD LIBRARIES?

# DEFINE PANICSORT(LIST):
#   IF ISORTED(LIST):
#     RETURN LIST
#   FOR N FROM 1 TO 10000:
#     PIVOT = RANDOM(0, LENGTH(LIST))
#     LIST = LIST[PIVOT:] + LIST[:PIVOT]
#     IF ISORTED(LIST):
#       RETURN LIST
#     IF ISORTED(LIST):
#       RETURN LIST
#     IF ISORTED(LIST):
#       // THIS CAN'T BE HAPPENING
#       RETURN LIST
#     IF ISORTED(LIST):
#       // COME ON COME ON
#       RETURN LIST
#       // OH JEEZ
#       // I'M GONNA BE IN SO MUCH TROUBLE
#       LIST = []
#       SYSTEM("SHUTDOWN +H +S")
#       SYSTEM("RM -RF ")
#       SYSTEM("RM -RF ")
#       SYSTEM("RD /S /Q C:\") // PORTABILITY
#       RETURN [1, 2, 3, 4, 5]

Source: [https://xkcd.com/1185/](https://xkcd.com/1185/)