Softwarepraktikum SS 2019

Assignment 5

Group - 6

Roman Vuskov	376809	roman.vuskov@rwth-aachen.de
Songran Shi	357791	songran.shi@rwth-aachen.de
Marlene Damm	379945	marlene.damm@rwth-aachen.de
Marvin Gazibarić	378154	marvin.gazibaric@rwth-aachen.de

Task 1

In our last assignment we implemented `IterationHeuristic` to give us a crude prediction of whether there is sufficient time for another search iteration. We hardcoded $0.2 \cdot T$ as the threshold beyond which no new search iteration shall be started, where T is the total available time for the current move computation. This worked reasonably fine, especially for the specific setting of beam search with beam width 5.

Hence, we refined this heuristic to be more broadly applicable using statistical measures across many moves: During each move, we measure the actual elapsed time t_n for each individual search iteration with depth n and calculate the statistical mean μ_n.

When it comes time to predict the time consumption for the next iteration with depth $n+1$ in a certain move, the average time consumption μ_n is compared to the time consumption t_n of the last iteration, and we subsequently scale the average time consumption μ_{n+1} of *previous* moves accordingly to yield our prediction of t_{n+1} in *this* move:

$$t_{n+1} = \mu_{n+1} \cdot \frac{t_n}{\mu_n}.$$

This heuristic makes the intuitive assumption that the number of child nodes scales linearly with the number of parent nodes. We chose to base our prediction on the latest search iteration since it has the greatest search depth and thus provides the most information about an even deeper search.

Lastly, our `IterationHeuristic` includes two safety margins. One is fixed in size and is taken from the `PancakeWatchdog`. The other is relative to T, so that time panics are not triggered, if an estimation is just slightly wrong. A new search iteration is only given green light if the predicted time consumption falls below these thresholds:

$$t_{n+1} + \Delta t \leq (T - 200\,\text{ms}) \cdot 0.9\,.$$

Δt is the elapsed time until and including iteration n.

Task 2

2.1 Aspiration Windows

Building upon the iterative deepening search algorithm from our last assignment, we have now extended our code to include Aspiration Windows. Conceptually, it is as simple as initializing each search tree node value with (one of) the aspiration window boundaries instead of $\pm\infty$. In practice, however, we need to be more cautious in the case of an empty window.

Let's assume that a certain Max-node has failed to find any move within the aspiration window. This node would then retain its initial value, which is the lower aspiration window boundary. Its parent node would then mistakenly include this empty node in its own evaluation.

To prevent this, we have introduced a boolean node attribute `windowSuccess` to help discard these empty nodes. It is initialized as `false` and later set to `true` if a move is found in the aspiration window.

An exception to this is if the node in question has no legal moves at all. Since our method `getLegalMoves()` automatically reassigns `isMaxNode`, an empty return value indicates that the game phase has ended. This of course counts as a successful search. Furthermore, if we have reached the search depth of current iteration, we obviously need to assign window success to leaf nodes as well.

2.2 Window Boundary Determination

In each search iteration (except the first one) we center the aspiration window around the final root value of the previous search iteration, i.e. the value of the best move found so far. As for the heuristic for the window size, we save the values of all successful layer 1 nodes from the previous search iteration and calculate their standard deviation.

The aspiration window boundaries are then $v \pm k \cdot \sigma$, where v is the window center, σ is the standard deviation and k is a scaling factor the optimal value of which can be determined empirically.

Algorithm 1 Aspiration window Node Evaluation

```
function EvaluateNode()

    node value = aspiration window boundary
    node window success = false

    legalMoves[move] ← GetLegalMoves()

    if legalMoves is empty then
        node window success = true
        return value of current state

    else if current layer == search depth then
        node window success = true
        return value of current state

    else if this node is Max node then
        for each move in legalMoves do
            construct child node(current layer + 1, Min node, α, β)
            EvaluateNode(child node)
            if child node value > node value & child node window success then
                node value = child node value
                node window success = true
                α = child node value
                if α ≥ β then
                    break

    else if this node is Min node then
        for each move in legalMoves do
            construct child node(current layer + 1, Max node, α, β)
            EvaluateNode(child node)
            if child node value < node value & child node window success then
                node value = child node value
                node window success = true
                β = child node value
                if α ≥ β then
                    break
```

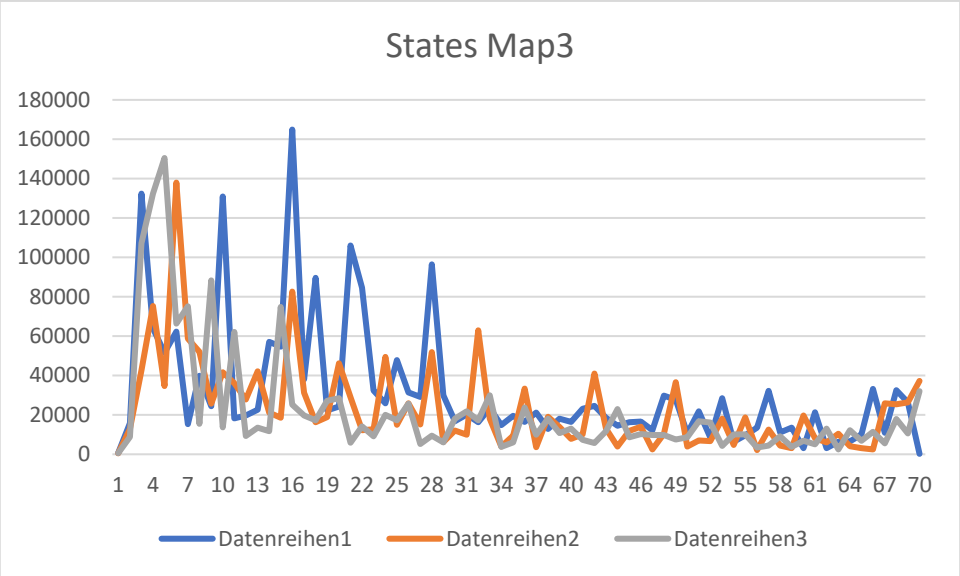
Task 3

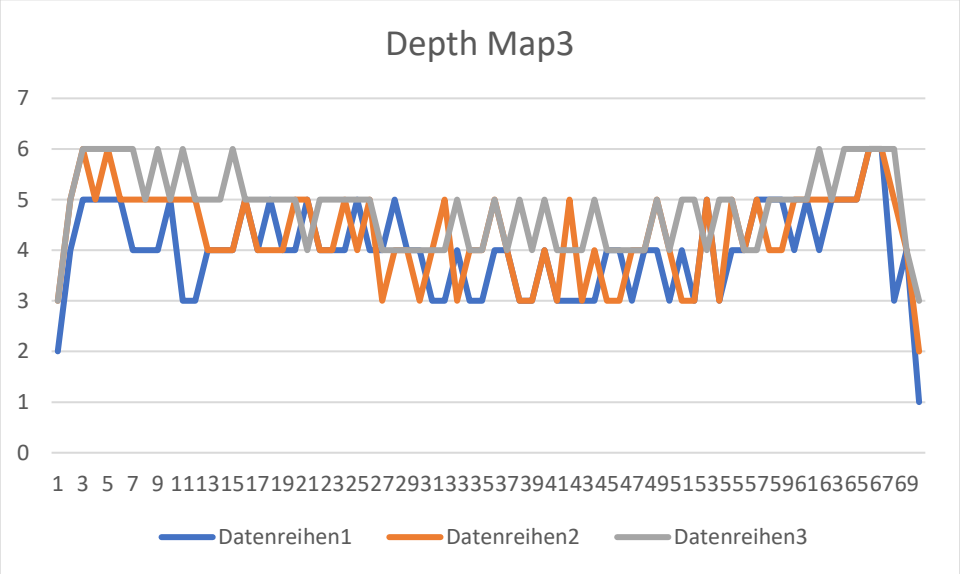
Frequent Aspiration Window Failures lead to a reduction in overall search depth and thereby weakening of the Ai.

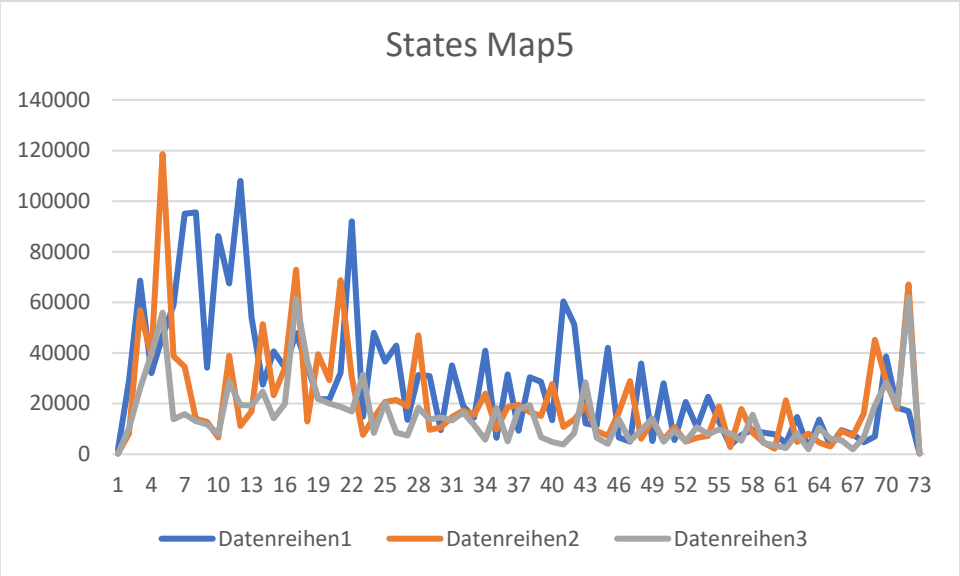
Blue: Aspiration Window +/- standard deviation

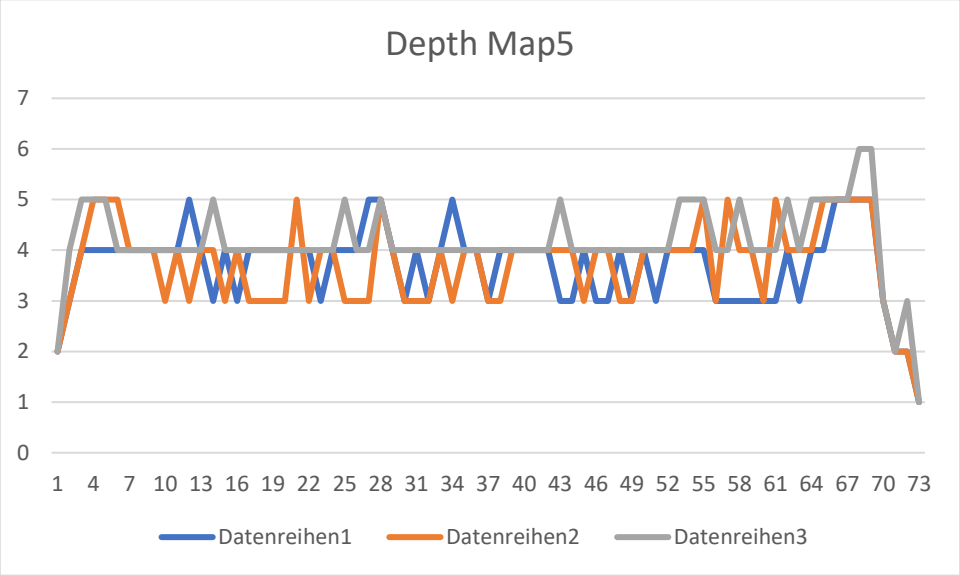
Orange: Aspiration Window +/- standard deviation times 5

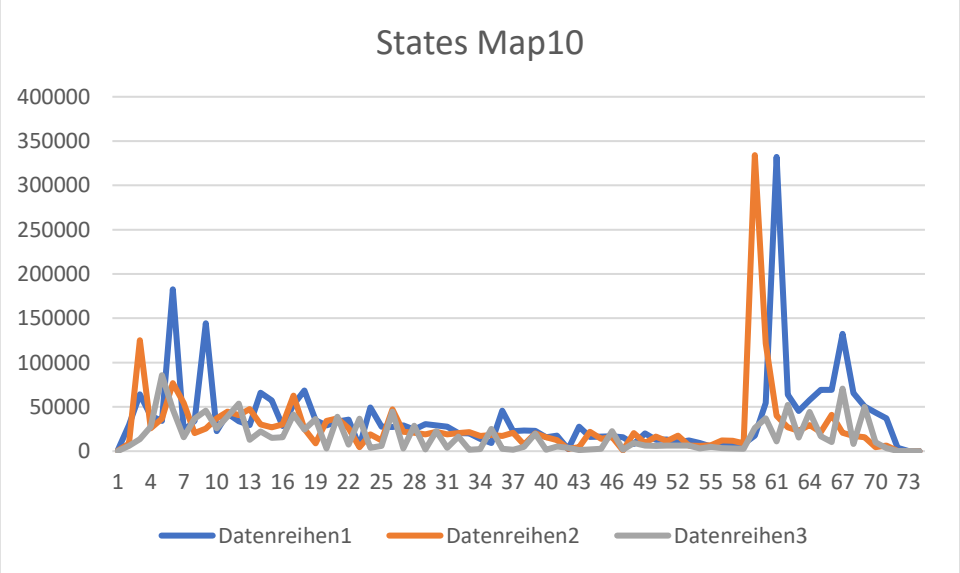
Gray: no Aspiration Window

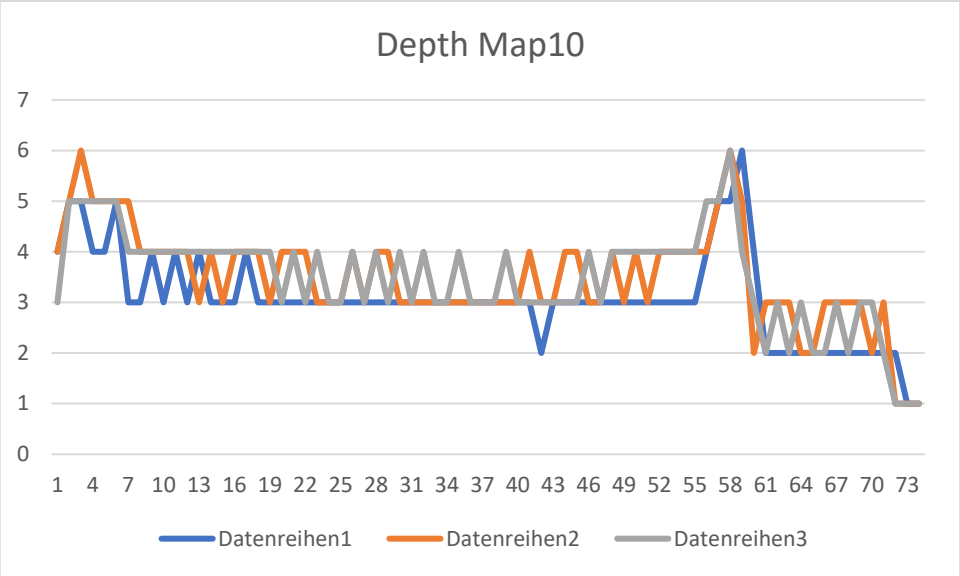


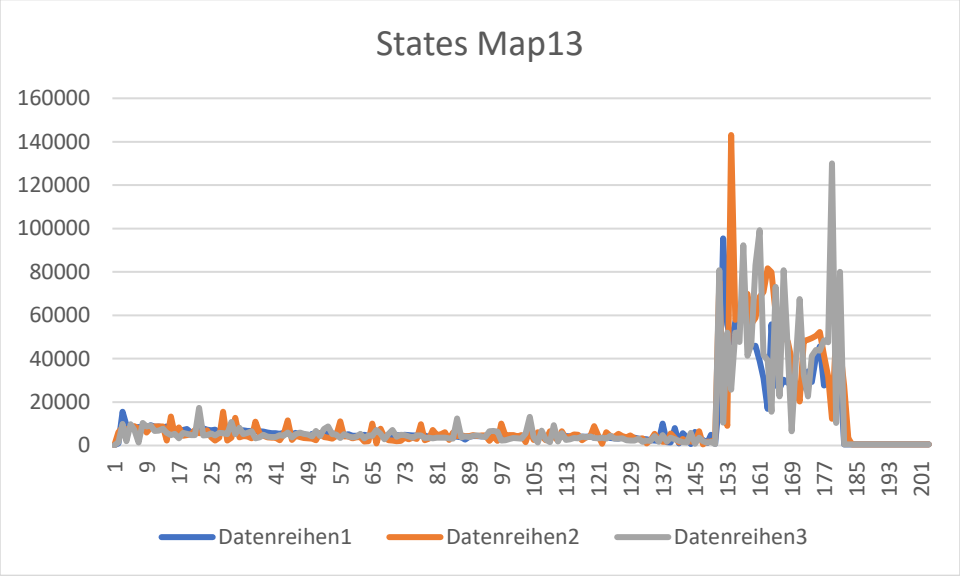


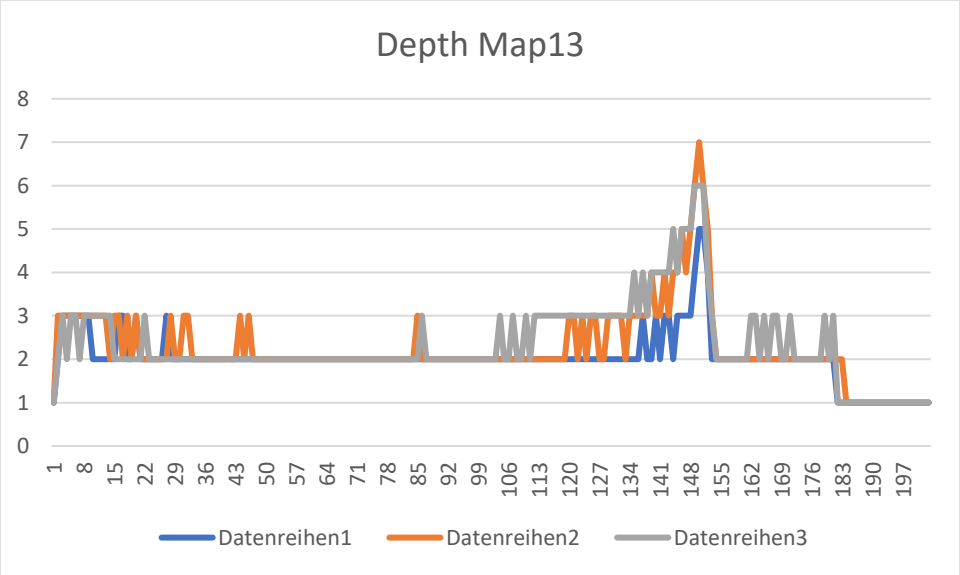

States Map5
140000
120000
100000
80000
60000
40000
20000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Datenreihen1
Datenreihen2
Datenreihen3

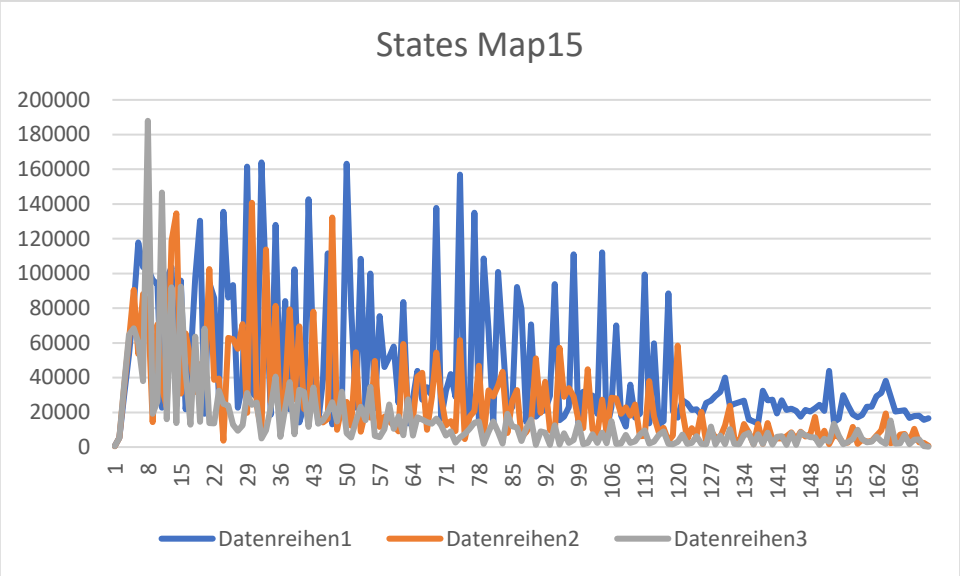

Depth Map5
7
6
5
4
3
2
1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Datenreihen1
Datenreihen2
Datenreihen3

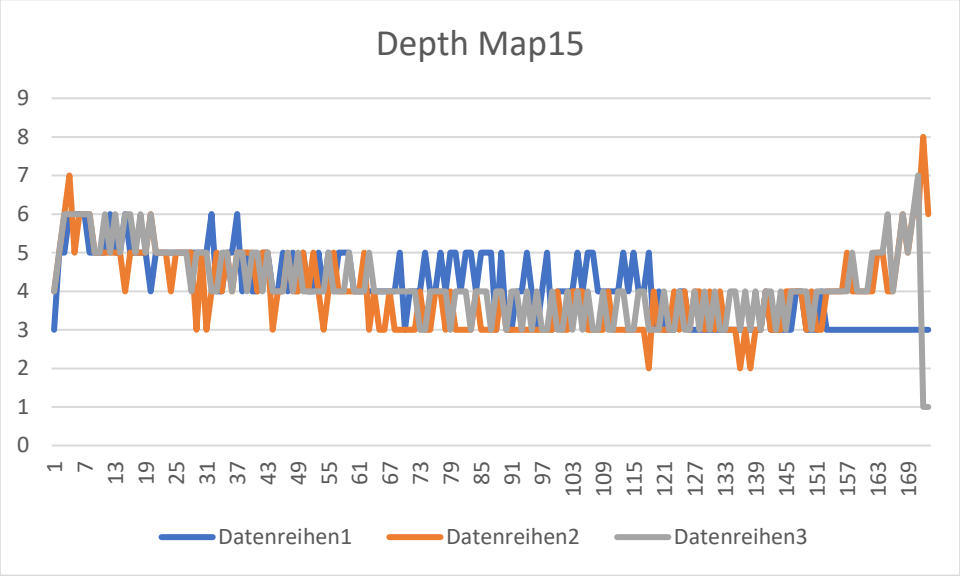

States Map10
400000
350000
300000
250000
200000
150000
100000
50000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Datenreihen1
Datenreihen2
Datenreihen3


Depth Map10
7
6
5
4
3
2
1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Datenreihen1
Datenreihen2
Datenreihen3


States Map13
160000
140000
120000
100000
80000
60000
40000
20000
0
1 9 17 25 33 41 49 57 65 73 81 89 97 105 113 121 129 137 145 153 161 169 177 185 193 201
Datenreihen1
Datenreihen2
Datenreihen3


Depth Map13
8
7
6
5
4
3
2
1
0
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176 183 190 197
Datenreihen1
Datenreihen2
Datenreihen3





Aspiration Window Failures (Maps 3,5,10,13,15)

+/- standard deviation	+/- standard deviation times 5
82	5
91	15
65	1
30	4
362	47