Softwarepraktikum SS 2025

Assignment 5

Group 6

Henri Schaudinn	456738	henri.schaudinn@rwth-aachen.de
Malte Ewald	456139	Malte.Ewald@rwth-aachen.de
Frithjof Alms	456247	frithjof.alms@rwth-aachen.de

Additional contents

1.1 Attribution

The time estimation algorithm required for task 1 was implemented and evaluated by Henri. The implementation and evaluation of the Aspiration Windows algorithm was done by Frithjof. Malte focused on task 4 and evaluated and improved our current heuristics. The part in the report was always written by the student who worked on the corresponding task.

Task 1

2.1 Estimation tradeoff

This task presents a unique challenge: accurately estimating the next depths calculation time is hard given the limited information and wide variance. We don't want to overshoot the calculation time since this would lead to a premature calculation stop. At the same time, underestimating the search time will lead to wasted time since a part of the calculation will now not contribute to the result.

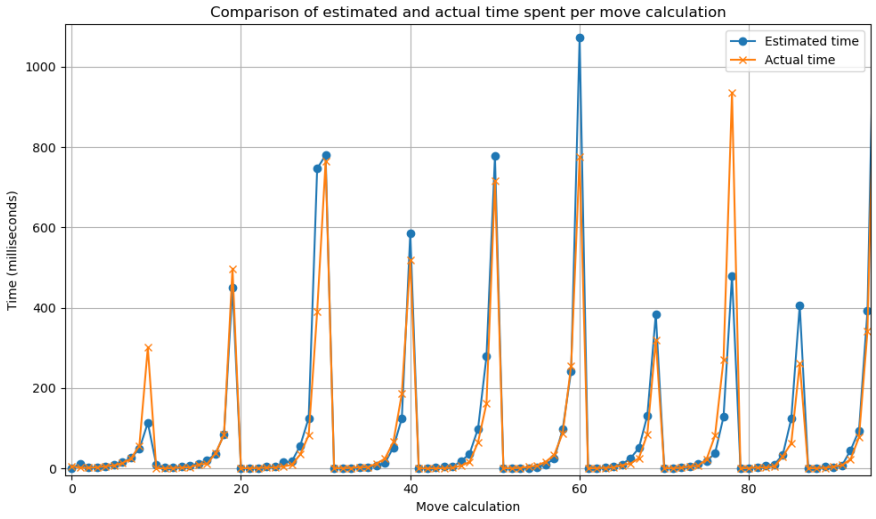

Figure 1: Time results on standard map

2.2 Estimation solution

To accurately estimate the next depths calculation time, we have different data to base our prediction on: firstly, there are the calculation times of the previous depths. These can be used to solve a regression problem that tries to predict the next point (next calculation time). Regression techniques can be quite accurate given limited information; on the other hand, calculations using more advanced regression techniques can be computationally expensive. Also, using regression in this way ignores a whole other data pot we can get access to: We can store the past move times for all individual depths, and that way, we have a very realistic expectation that even adjusts with the game phase and map. The results of this solution can be seen above. It can be observed that the graph is "periodic", which makes sense since when a move is finished and the calculation for a new move

starts. In the figure above, we can see that the time predictions for the first move stand out as significantly less accurate than the rest. Why is this? Since we are usually using the last move's time, and this is the first move, there is no last time we can fall back on. Therefore, in this first step, we deploy a simple regression strategy that works by approximating the line based on the assumption of exponential growth. We can also measure the accuracy by determining the rate of how often a move calculation is canceled by deadline expiration versus time estimation. It turns out our strategy gets close to a 60 percent success rate, but this success rate is of course not set in stone, in fact we could easily get it to 100 by just scaling the curves by a constant factor (we got the shape right already we might just want to make it bigger). The reason why we are not doing this right now is that with every bit we underestimate less, we will overestimate more. This tradeoff is worth it to explore more in the future, but it is essential to do long-term testing on a wide variety of opponents and maps, since extra time in the end might not be much of a use on maps where there is a lot of bombing or an end through the full map. On the other hand, if the possible moves stay high until the end (for example, on maps with lots of players, you could go from a lot to zero possible moves in just one round), it is very advantageous to have extra time in the end.

Task 2

The Aspiration Windows algorithm, which works on top of the deepening search algorithm, requires an additional parameter specifying the default window size, called *AspirationSize*. The move search on depth 1 runs similarly to the simple deepening search algorithm, but the heuristic score of the resulting move is saved. For any iteration with a depth limit greater than one, the algorithm is called with an initial alpha value of $H - AspirationSize * |H|$ and a beta value $H + AspirationSize * |H|$, where H is the heuristic score of the move found in the previous iteration. Additionally, the invoked algorithm does not perform cutoffs in the function instance called with the original depth limit. If the search fails, indicated by a returned move value of -1, the algorithm is called again using the minimum and maximum double values as alpha and beta. If the Aspiration Windows algorithm is called with an *AspirationSize* of zero, it performs a normal deepening search using no initial alpha and beta values.

Task 3

4.1 Test structure

To efficiently test different sizes of the aspiration window , we set up a testing function running the Aspiration Windows algorithm using different double values as *AspirationSize*. It tests all multiples of $0,05$ between zero and one. Values greater than one were not included in the test, as the Aspiration Windows algorithm performs similarly to the standard deepening search algorithm for such large windows. In every move of a game, the Aspiration Windows function was called using each *AspirationSize* and a consistent depth limit as parameters. The total execution time and the total number of states visited were counted for each function call.

4.2 Results

As proposed in the assignment, we conducted the test on several different maps. To display the test results, we chose a consistent diagram format seen in figure 2. Because we only tested multiples of 0.05 as *AspirationSize*, the numbers on the horizontal axis are the corresponding factor, meaning, for example, the measured values for an *AspirationSize* of 0.5 are displayed above the number 10 on the horizontal axis. The map used in 2 has a length of 15 and a width of 15, and there are no choice tiles and inversion tiles on the map. The optimal *AspirationSize* on this map is between $0,25$ and $0,3$, since the average amount of states visited per move and the total computation time per move reaches its minimum. However, values as *AspirationSize* smaller than 0.25 perform generally poorly on all tested maps, as the amount of restarted searches and therefore checked states and computation time are the highest.

In 3, the games were played on a map with a length and width of 20, offering choice and inversion tiles. The optimal *AspirationSize* here is between $0,3$ and $0,4$, relatively close to the optimum found in 2. On the larger tested maps, seen in 4 and 5, both having a length and width of 50 tiles, the optimal *AspirationSize* is around $0,5$ to $0,55$, while the optimal *AspirationSizes* found in earlier test perform rather poorly on the larger maps. Furthermore, the choice tiles on the pikatchu map did not affect the optimal *AspirationSize*, since the chess map does not have choice or inversion tiles and has a similar optimal *AspirationSize*.

4.3 Conclusion

The optimal *AspirationSize* is heavily dependent on the map size as analyzed in the previous section, so we decided to set the size of the $AspirationWindow$ to

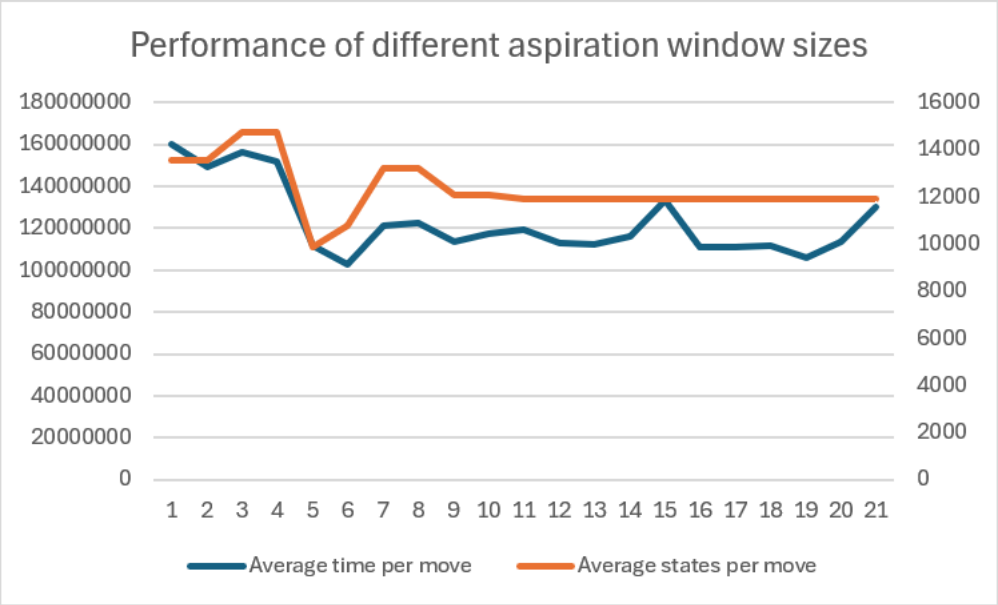

Figure 2: Testing results on the map "2025_g8_1_trans.MAP". The left vertical axis shows the average execution time per move in nanoseconds. The right vertical axis shows the average number of states visited during each move. Both of these values are calculated for every multiple of 0.05 between 0.05 and 1, the horizontal axis shows the corresponding factor. The twenty-first data point displays the measurements recorded using the Iterative Deepening algorithm without an aspiration window.

$(Width * Height)/10000 + 0,275$, as this formulas result is in the optimal range for every tested map. The results of these test are currently not used, as the tests were conducted using the heuristic "countAllHeuristicBase", because task 4 for this assignment was not yet completed. After we settle on a final heuristic, which will also be used for matches on Matchpoint, we will need to repeat these tests using the new heuristic, as it can influence the optimal $AspirationSize$. Secondly, we will try to repeat the test with higher depths, as, for example, the depth limit of six used in 4 led to rather small average times spent per move (less than 3% of the one second used on Matchpoint). This resulted in inaccuracies, a search with an $AspirationSize$ of $0,95$ took on average more than twice the time as a search with an $AspirationSize$ of $0,85$, while the number of checked states is identical. Another problem, which we could not explain, is the high average computation time of the standard iterative deepening search, seen above the number 21. It may be caused by cpu optimization, as it was called before the Aspiring Windows algorithm. After we switched both function calls, the Aspiring Windows algorithms' average computation time rose, while the average computation time of the

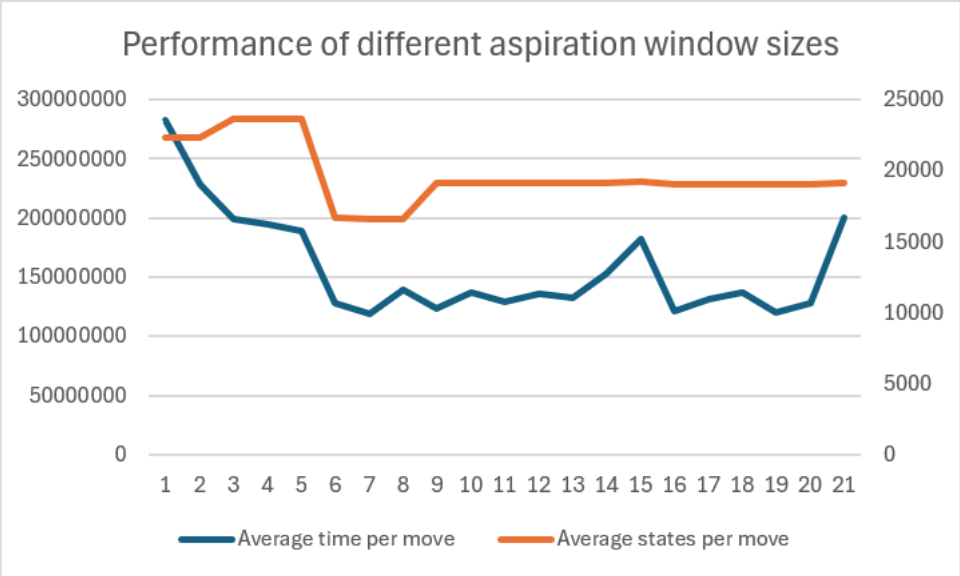

Figure 3: Testing results on the map "2025_g6_3_competetive.MAP". The diagram specifications for the horizontal and vertical axis and units of measurements are identical to the ones used in 2.

standard iterative deepening search sunk to an expected value.

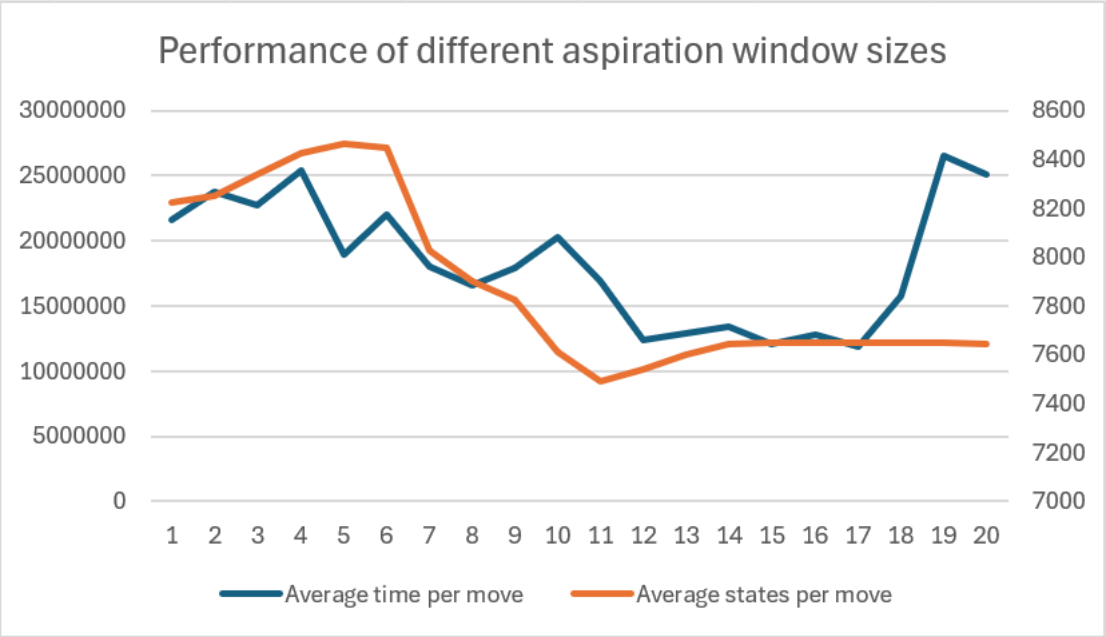

Figure 4: Testing results on the map "2025_g4_1_chess.MAP". The diagram specifications for the horizontal and vertical axis and units of measurements are identical to the ones used in 2.

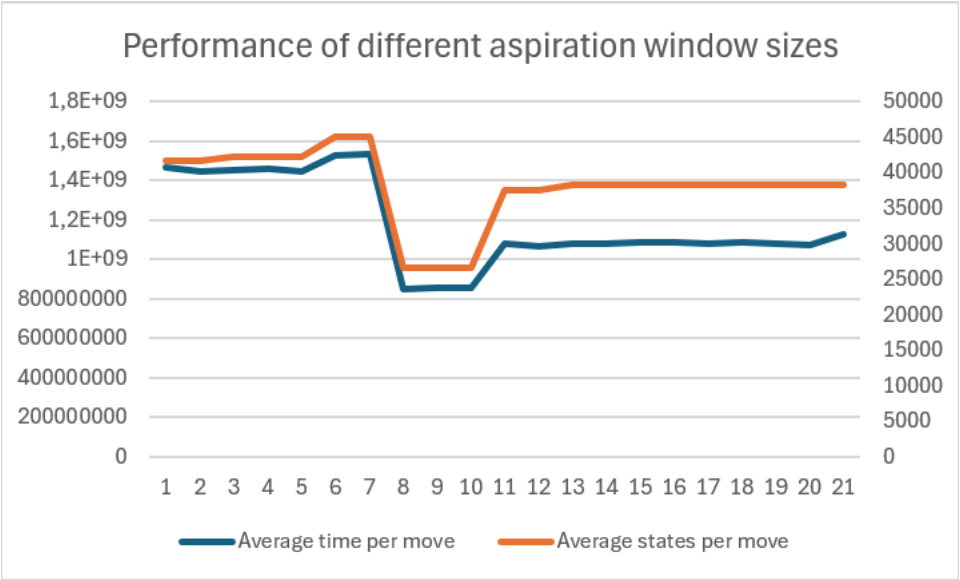

Figure 5: Testing results on the map "2025_g6_2_pikatchu.MAP". The diagram specifications for the horizontal and vertical axis and units of measurements are identical to the ones used in 2.

task 4

5.1 what predicts victory?

In the earlier part of the project, we started implementing different heuristics. These heuristics tried to predict the outcome of the match based on different factors. These factors were: number of stones, number of cornerstones, number of overrides, number of bombs, inversion stones left.

In the first step, we will revisit these factors and examine the correlation between them and victory.

5.2 Circle Map corner vs count all

I had a look at 8 games on the matchpoint server and had a look at how the number of stones and corners correlate with victory. I measured at the half and three-quarter mark, and the end of the game.

At the end, count all, of course, had a correlation of 1 with victory. On the other hand, corners only correlated 0,875 because of one statistical outlier. This outlier score was 592|602 before bombs, while corners were 21|19. So it was very close.

This renders the significance of this outlier rather small.

At the three-quarter mark, the same was true, but the deltas between the two count all results became smaller while the differences between the number of corner stones stayed quite high.

At the halfway mark, the situation changed. Corners still had a correlation of 0,875 with the exception of the one outlier discussed before. On the other hand, count all only had a correlation of 0,5, which would mean no correlation. This is probably a statistical error because of the low number of values looked at. But it does show a trend.

With this, I make the first three working hypotheses: At some point between the half and three-quarter mark of the game count all rises in significance. At the start, it is rather insignificant. Until that point, corners have a much higher correlation with victory. Corners have a high correlation with victory if no special stones are involved.

5.3 inversion/swap problem

The results from the last section only play a role in games, without inversion/swap stones, as inverse stones invert the game, thereby reversing the meaning of the heuristic.

The simple workaround would be to just assign every player a "relative player " number, which is the player whose stone they will receive after all inversions are done.

This is implemented in our new improved version. But it faces two glaring flaws.

1. Not every inversion space has to be taken. 2. If the amount of stones owned by you fall below a certain threshold you can be forced to use override stones, which affects your chances of victory very negatively. More on that later.

Another problem is that we still do not have a handle on swaps, because we cannot predict what is going to happen with them involved.

5.4 bonus tiles multiplayer

When looking at the correlation of the results of the Four Map to the number of bonus Tiles occupied, we find a significant correlation. I had a look at eight games, and these were the results.

Table 1: correlation bonus placement

placement	1	2	3	4
total	24	9	5	2
relative	0,6	0,23	0,13	0,05

As we can see, the amount of bonus tiles occupied is highly correlated with the placement of the player on this map. But why is that?

5.5 stockpiling overrides

The reason is an imbalance in the amount of tiles on the map relative to the number of override stones. This gives a player the ability to capture a lot of the map at the end of the game if they stockpile their override stones.

And if a player does that, the rest of the players on that map will almost surely lose if they do not do the same. This causes the "meta" to shift because the results of the game until the override stones start being placed do not hold much significance on these maps. So in the first part of the game, the only significant tiles become the bonus tiles because they can give you more overrides.

This is only significant on maps with this imbalance. But in our map pool, these maps are quite prevalent.

5.6 Improvements we made

1. Inversion Idea as in 5.3 2. We gave overrides a very high value, so they do not get used until the very end. This also causes bonus tiles to be valued very highly 3. We valued corners higher than normal stones.

5.7 results of testing

results on 20/06/2025, 08:06:22

	Disq.	1st	2nd	3rd	4th	5th	6th	7th	8th
2 Player Map	0×	3×	13×						
3 Player Map	0×	0×	8×	4×					
4 Player Map	0×	1×	5×	7×	7×				
5 Player Map	0×	0×	0×	1×	2×	2×			
6 Player Map	0×	0×	1×	2×	3×	0×	0×		
7 Player Map	0×	0×	0×	0×	0×	0×	0×	0×	
8 Player Map	0×	0×	5×	6×	9×	5×	13×	15×	3×

As we can see, our results were rather bad overall. But the results on the multiplayer maps were especially bad. After our improvements, the results looked like this:

results on 22/06/2025, 10:08:10

	Disq.	1st	2nd	3rd	4th	5th	6th	7th	8th
2 Player Map	0×	5×	11×						
3 Player Map	0×	3×	6×	3×					
4 Player Map	0×	2×	2×	5×	11×				
5 Player Map	0×	1×	0×	1×	0×	3×			
6 Player Map	0×	1×	1×	1×	1×	1×	1×		
7 Player Map	0×	0×	0×	0×	0×	0×	0×	0×	
8 Player Map	0×	6×	9×	6×	12×	6×	12×	4×	1×

As we can see, the results of our ai improved significantly, especially on the 8-player maps. This can mostly be attributed to the changes made in the weighting of the override stones. Problems with specific maps and the 2-Player maps remain.

5.8 outlook

There are still improvements to be made here, as sometimes the strategy of holding overrides causes us to simply lose the game because we have no more moves. Or we get forced to expend overrides, because we have no more moves, so tracking the number of possible moves and including them in the heuristic could be very helpful for this case.

Also, a dynamic weighting of the heuristics in regard to corners vs. total stones based on the progression of the game could be envisioned. Here we face a problem concerning the previously mentioned amount of possible turns, because there is a correlation between the possible turns and the number of stones one controls. So we would first need to implement the previously mentioned idea to make this viable.

On the matter of efficiency, our heuristics currently do not significantly constrain our efficiency. This can be seen in the previous report.