— GUIDE —

Assignment 2: Play FAUhalma

AI-1 Systems Project (Summer Semester 2026)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

This document is intended to help you solve the assignment "Assignment 2: Play FAUhalma" [AS]. You do not have to read it, but we do recommend to at least take a look at the tips and common issues.

1 A few tips

1. Make sure you have a good representation for FAUhalma positions.
2. Make sure you get all legal moves (and only legal moves) for a FAUhalma position (maybe implement unit tests).
3. Consider implementing and testing very simple agents before moving to more sophisticated algorithms.
4. You can use the server to see how well your agent performs and visualize what moves it chose. Nevertheless, it might be useful to have some tools to analyze and compare different agent implementations locally. As long as you don't share agent implementations, we don't mind if you collaborate with other teams on that.

2 Brief consideration of different approaches

1. **Greedy.** If you implement a greedy agent with a simple heuristic, your agent should already perform well in the easy environments and get many points. This is also a good starting point before you move to the more advanced algorithms as it lets you test if the agent finds the correct moves (and if the heuristic works). In the past, some students got really impressive results with a greedy agent and a sufficiently sophisticated heuristic.
2. **MCTS.** Monte Carlo Tree Search can be used to solve this assignment. Some of the strongest agents in the past used it. However, subtle bugs in the implementation can

make the agent perform poorly and finding these bugs can be difficult (though this is also true for the minmax-based algorithms).

3. **Minmax-based**. Minmax-based algorithms can also work well for this assignment, but the high branching factor of the game can be a challenge. Applying it to 3 players requires some ingenuity (especially if you use alpha/beta pruning). One option is to pretend that the opponents are a single player. There are many variations of this. One variant is that the opponent moves either an A peg or a B, but not both. Another variant is that the opponent moves both an A and a B peg. In this variant, it can also make sense to restrict the number of combinations by considering one of the pegs greedily (i.e. combining all A moves with the greedy B move and vice versa).

3 Common Issues

1. Considering hop chains with the same start and target position as different moves if they have different in-between positions. This can substantially increase the branching factor.