

# Assignment 2: Play FAUhalma

AI-1 Systems Project (Winter Semester 2022/2023)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic: Adversarial Search

Due on: January 17, 2022 (*early track*: December 20, 2022)

Version from: November 28, 2022

Author: Jan Frederik Schaefer

## 1 Task Summary

Implement an agent for a variant of the game Sternhalma/Chinese checkers [CC], which we will call FAUhalma. For evaluation, your agent will compete on our server at <https://aisysproj.kwarc.info>.

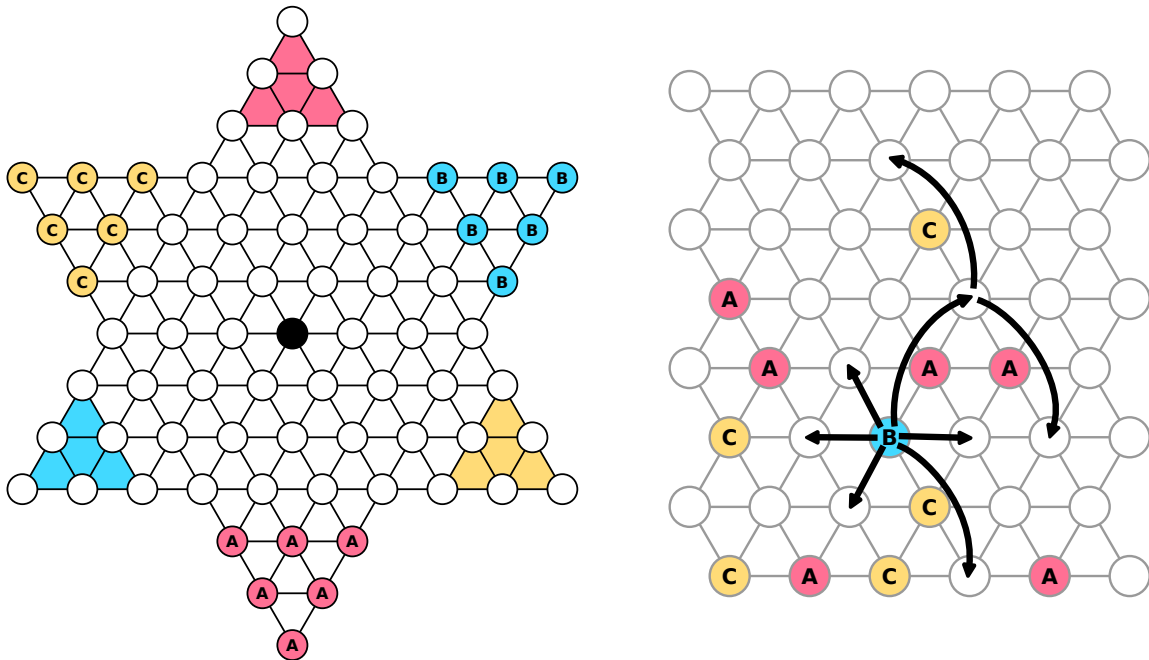


Figure 1: Starting position for a game of FAUhalma with players *A*, *B* and *C* (left) and an illustration of the move rules (right).

## Objectives

1. Gain some experience with implementing and adapting algorithms for adversarial search,
2. learn how to work with a non-rectilinear grid,
3. get to know the JSON format (if you do not know it already).

## Prerequisites and useful methods

1. Adversarial search (as discussed in the AI lecture),
2. search in general.

## 2 Rules of FAUhalma

**FAUhalma** is a 3-player game, modified from the original game Sternhalma/Chinese checkers [CC] for AISysProj. Like the original, FAUhalma is played on a star-shaped board. Each player has several **pegs**, which originally reside in the player's **starting corner**. The goal of each player is to move their pegs into the opposite corner (the **home**) as quickly as possible. Figure 1 illustrates the starting position and the homes (shaded areas). Players take turns counterclockwise. When it is their turn, a player moves one of their pegs by either moving it to an adjacent space or by hopping over other pegs (see Figure 1). Here are the move rules in more detail:

- **Simple move:** Move the peg to an adjacent empty space (in any direction).
- **Simple hop:** If there is a peg on an adjacent space  $S$ , the peg can “jump over it” as long as the space behind  $S$  is empty.
- **Hop chain:** A chain of simple hops; i.e. if a peg can hop from  $S_1$  to  $S_2$  and it could then hop from  $S_2$  to  $S_3$ , then it can also hop from  $S_1$  to  $S_3$  in a single move. However, at the end of a chain, the peg must land on a different space than it started from. Note that a simple move cannot be combined with hops.
- **Swap rule:** Spaces in the home of the moving player that are occupied by an opponent's peg can also be considered empty for the rules above (except for the intermediate spaces in a hop chain). If a peg is moved from  $S_1$  to  $S_2$  and  $S_2$  is occupied by an opponent's peg, then the peg from  $S_2$  is moved to  $S_1$ , i.e. the pegs are swapped. The swap

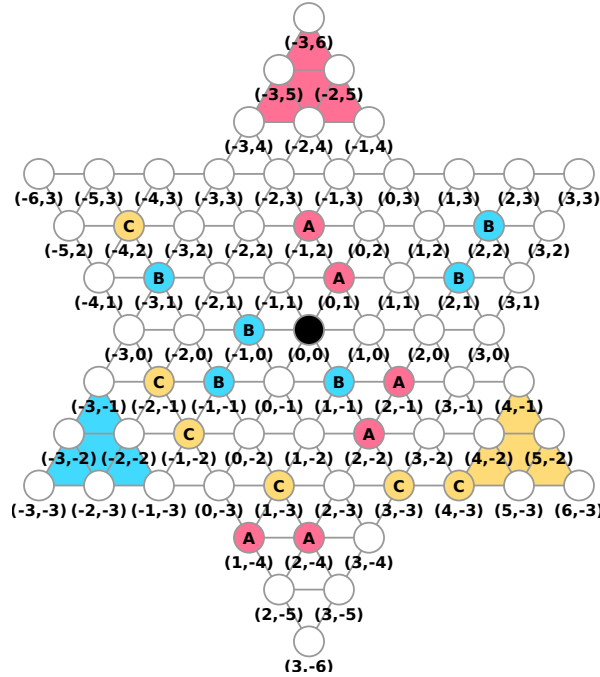


Figure 2: Coordinate system for FAUhalma.

rule prevents players from blocking each other by moving pegs into their opponents' homes.

- **Center rule:** You can not move a peg onto the space in the center (it is blocked), but you can hop over it like over pegs.

It is theoretically possible that a player has no legal move. In that case the blocked player loses. The first player who has moved all their pegs into their home wins. The remaining players continue to determine who gets the second place.

### 3 Coordinates, Positions and Moves

To test your agents, we need to have a standardized way of sharing moves and positions, which will be discussed in this section.

#### 3.1 Coordinate System

We will refer to the spaces using a coordinate system. Figure 2 illustrates what coordinates correspond to which space. As you can see, every space is associated with a coordinate

pair  $(x, y)$ . However, the  $x$ -axis and  $y$ -axis are not orthogonal in the normal visualization. Working with such grid like that can be somewhat tricky. Maybe you can find an elegant way?

## 3.2 Representation of Positions

The server sends you the position (game state) as a JSON object, indicating where each agent has its pegs. For example, the position shown in Figure 2 would be represented as

```
{"A": [[2, -2], [-1, 2], [2, -4], [2, -1], [1, -4], [0, 1]],  
  "B": [[2, 1], [-1, 0], [2, 2], [-1, -1], [1, -1], [-3, 1]],  
  "C": [[3, -3], [-1, -2], [4, -3], [-4, 2], [1, -3], [-2, -1]]}
```

Your agent is always player *A*.

## 3.3 Representation of Moves

Moves are represented as a JSON list of coordinate pairs, where the first coordinate pair indicates the peg you want to move and the last one indicates what space you want to move the peg to. If you have a hop chain, the list has to include all intermediate spaces, i.e. two consecutive list entries should be a simple hop apart from each other. For example, these would be valid moves for player *A* in Figure 2:

- `[[2, -2], [2, -3]]`
- `[[1, 2], [1, 0], [3, -2], [3, -4]]`

## 3.4 Simplified variants

The server also has simplified variants of FAUhalma, using either the star-shaped board of FAUhalma or a rhombic board, which is basically a star-shaped board with some corners cut-off (Figure 3). The variants have the same coordinate system and move rules as the standard variant of FAUhalma.

# 4 Evaluation: Playing on the Server

Your agent will be evaluated by playing on our server: <https://aisysproj.kwarc.info>. In a two-player game, your agent will get 1 point if it finishes first and 0 points otherwise. In a three-player game, your agent will get 2 points for finishing first, 1 point for finishing

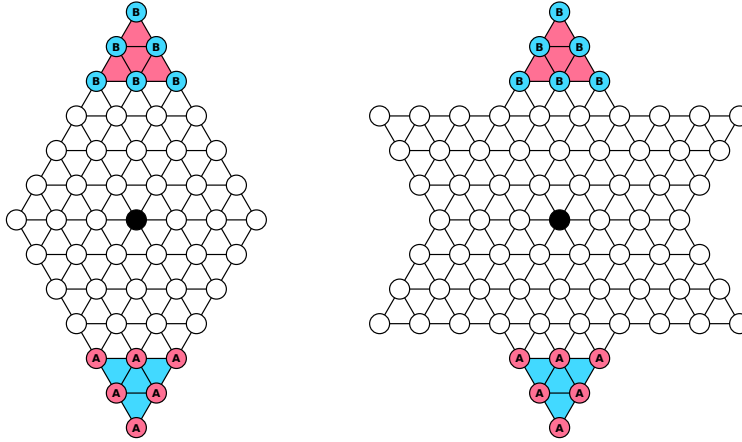


Figure 3: The starting positions of simplified FAUhalma variants.

second and 0 points for finishing last. Your agent will be rated by taking the average points of 50 consecutive games. The server remembers your best rating of 50 consecutive games.

The server has several environments for evaluating your agent with differing settings. See Section 7 for more details.

You can see your agent’s ratings and games on the web interface of <https://aisysproj.kwarc.info>.

## 4.1 Credentials and Client Implementations

For each environment on the server, your team repository will contain one configuration file with credentials that you can use to test your agent in that environment. The assignment repository [AR] contains an example implementation of the server protocol in Python. Essentially, all you have to do is implement a function that, given a position, returns a move. If you would like to use a different programming language, you are of course welcome to implement the protocol (see Appendix A) yourself and we will try our best to support you. In that case, it would be nice if you share your implementation so that other/future students can use it as well.

## 5 What to submit

Your solution should be pushed to your gitlab repository for this assignment. Concretely, the repository should contain:

1. all your code for solving this assignment,
2. a README.md file explaining how to run your code (including e.g. dependencies that have to be installed),
3. a brief summary of how you solved the problem either as a PDF file ( $\approx 1$  page) or as part of your README.md.

## 6 Random 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.

## 7 Points

The total number of points for this assignment is 100. You can get up to 20 points for the quality of the submission (README, explanation, ...). Furthermore, you can get up to 80 points for the strength of your agent. Each environment allows you to get a certain numbers of points if your agent is strong enough. When grading, we will only consider the environment in which you would get most points (we do not add up results from different environments). That means that you do not have to run your agent on every environment and you can get full points if you only run it on the most difficult one where you can get 80 points.

Note: It is okay if your agent is a bit “lucky” and gets a slightly higher rating than it usually does. We will use the best rating on the server for your grade, assuming that we can reproduce a similar performance (i.e. if you get a 1.03 rating on the server and we get a 0.94 rating that is okay – if you get a 1.03 rating on the server and we get a 0.31 rating when testing it, we might ask some questions).

Config file	Shape	Players	Points
ws2223-2.1-rhombus.json	rhombus	2	30 if rating > 0.5
ws2223-2.2-easy-2-player.json	star	2	40 if rating > 0.5
ws2223-2.3-medium-2-player.json	star	2	55 if rating > 0.5
ws2223-2.4-difficult-2-player.json	star	2	65 if rating > 0.5
ws2223-2.5-easy-3-player.json	star	3	60 if rating > 0.5; 70 if rating > 1.5
ws2223-2.6-medium-3-player.json	star	3	75 if rating > 1.0
ws2223-2.7-difficult-3-player.json	star	3	77 if rating > 1.0
ws2223-2.8-hardcore-3-player.json	star	3	80 if rating > 1.0

## References

- [AR] *Repository for Assignment 2: Play FAUhalma*. URL: <https://gitlab.rrze.fau.de/wrv/AISysProj/ws2223/a2-play-fauhalma/assignment>.
- [CC] *Chinese checkers*. URL: [https://en.wikipedia.org/wiki/Chinese\\_checkers](https://en.wikipedia.org/wiki/Chinese_checkers) (visited on 11/10/2021).

```

// First request:
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkl2WmBDldsYHD3wLwUQAKoG_2_xBcE", "actions": []}

// First response
{"errors": [], "messages": [], "action-requests": [
  {"run": "40#1", "percept": ...},
  {"run": "7#3", "percept": ...}]}

// Second request
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkl2WmBDldsYHD3wLwUQAKoG_2_xBcE",
  "actions": [{"run": "40#1", "action": ...},
              {"run": "7#3", "action": ...}]}

```

Figure 4: Example interaction with the server.

## A The AISysProj Server Protocol

For some assignments, agents are tested by interacting with the AISysProj server. This appendix describes the protocol and is **only relevant if you want to implement the server protocol yourself**.

The protocol is rather minimal: You send an HTTP request to the server with your credentials and actions, and the server responds either with an error message or with a list of action requests that you should respond to in the next request. For the first request, you will have to send an empty list of actions to get action requests from the server (see Figure 4 for an example).

**Configuration files** The server details and your credentials are stored in configuration files. Usually, we will generate configuration files for you and commit them to your repository. They are JSON files that contain the following fields:

- **url**: the server URL,
- **env**: the environment,
- **agent**: the agent's name, and
- **pwd**: the agent's password.

**The request** You should send a PUT request to `[url]/act/[env]`, where `[url]` and `[env]` are provided by the configuration file. The request body should contain a JSON object with the fields:



- **agent**: the agent's name (from the configuration file).
- **pwd**: the agent's password (from the configuration file).
- **single\_request**: **true** or **false**, indicating if a only a single action request should be sent (can be helpful for debugging).
- **actions**: the actions the agent wants to do as a list. Each action is represented as an object with two fields: **run** is an identifier of the action request (provided in the server responses) and **action** is the action the agent wants to do – the value format depends on the assignment.

**The response** If the request was accepted, you will receive a JSON response with the fields:

- **action-requests**: a list of action requests that you should send actions for in the next request. Each action request is an object with two fields: **run** is an identifier so that your action can be linked to the request and **percepts** describes what you know about the current state (e.g. the position in a game).
- **errors**: a list of error messages (e.g. if your move was invalid).
- **messages**: a list of other messages.

**Error response** In case of an error (e.g. invalid credentials), you get a JSON response with the fields:

- **errorcode**: The HTTP error code.
- **errorname**: The name of the error.
- **description**: A more detailed description of the error.