Problem 2: Implement an Agent for FAUhalma

AI1SysProj 2021

Topic:	Adversarial Search
Due on:	January 18, 2021
Version from:	December 2, 2021

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 against other agents.

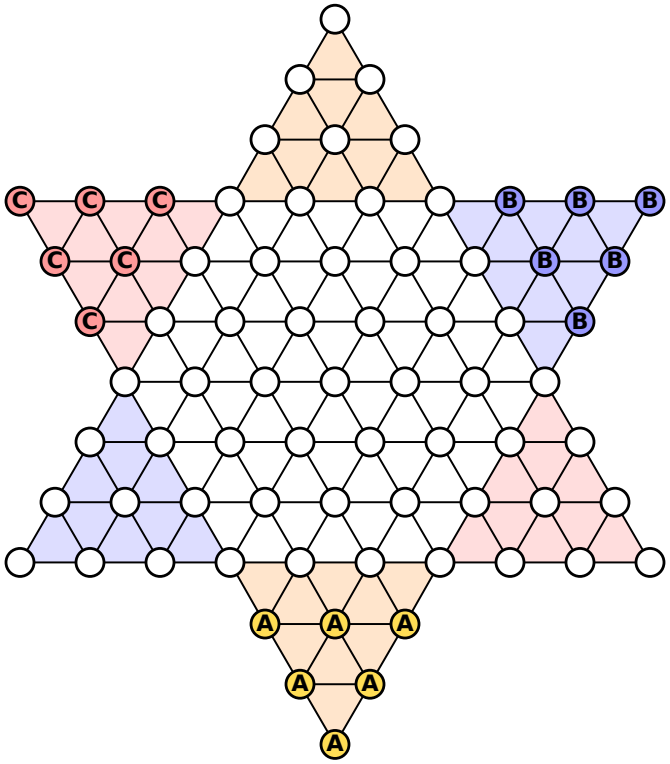

Figure 1: Starting position for a game of FAUhalma with players A, B and C. Player A has the first turn, followed by player B and then player C.

2 Rules of FAUhalma

FAUhalma is a 3-player game, simplified from the original game Sternhalma/Chinese checkers [CC] for AISysProj. Like the original, FAUhalmais played on a star-shaped board. Each player has 6 **pegs**, which originally reside in the player's **starting corner**. The goal of each player is to move the pegs into the opposite corner (the **home**) as quickly as possible. Note that only the 6 outermost spaces form the home. Figure 1 illustrates the starting position. 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 2). 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 rule prevents players from blocking each other by moving pegs into their opponents' homes.

It is theoretically possible that a player has no legal move. In that case the game is abandoned.

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 let different agents compete with each other, we need to have a standardized way of sharing moves and positions, which will be discussed in this section.

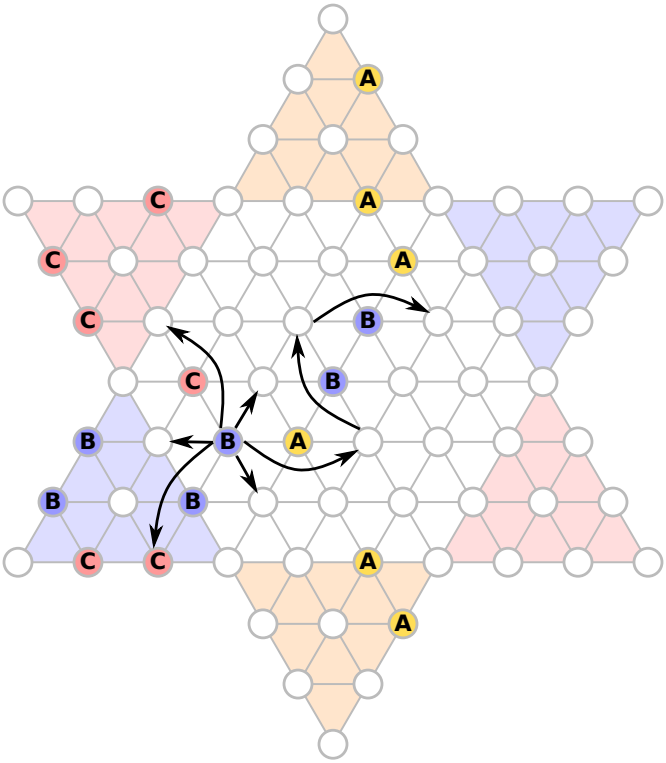

Figure 2: All possible moves for one peg of player B.

3.1 Coordinate System

We will refer to the spaces using a coordinate system. Figure 3 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 and we don't have a rectilinear grid. Working with a grid like that can be somewhat tricky. Maybe you can find an elegant way?

3.2 Representation of Positions

The server sends a string representation of the positions (game states) you should provide a move for. Positions are represented in the following form:

```
<YOUR PEGS>:<PEGS OF NEXT AGENT>:<PEGS OF LAST AGENT>
```

Recall that agents take turns counterclockwise. The pegs of an agent are represented as a semicolon-separated list of the peg coordinates. Coordinates are represented as

```
<X COORD>,<Y COORD>}
```

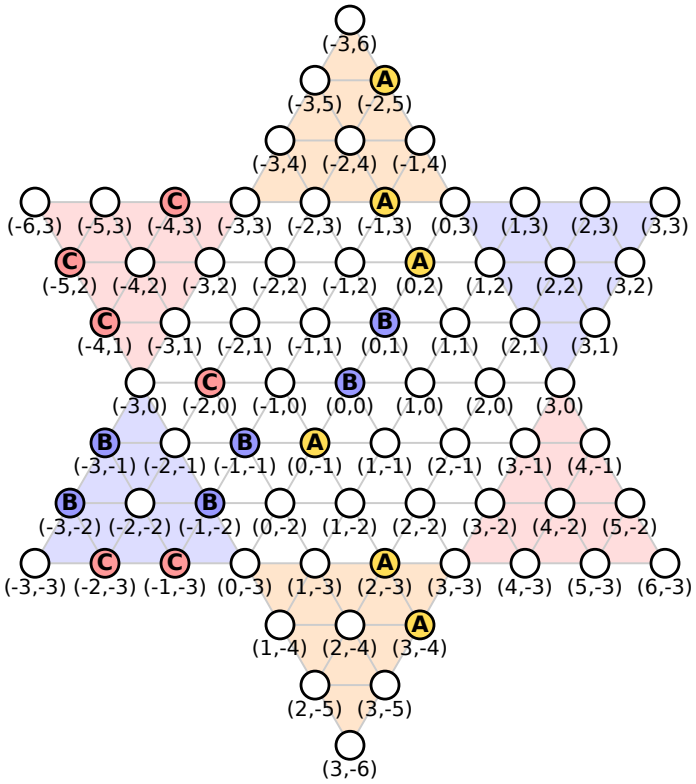

Figure 3: Coordinate system for FAUhalma.

For example, the position shown in Figure 3 would be represented by the following string (without the line breaks):

```
-2,5;-1,3;0,-1;0,2;2,-3;3,-4:
-3,-2;-3,-1;-1,-2;0,0;0,1;-1,-1:
-5,2;-4,1;-4,3;-2,-3;-2,0;-1,-3
```

Note: For simplicity the board is always rotated such that your home would be the top corner of the board.

3.3 Representation of Moves

A move is also represented as a string. Concretely, it is a semicolon-separated 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 3:

- `0,-1;-2,-1;-2,1`
- `3,-4;3,-3`

4 Evaluation: Playing on the Server

We can evaluate the strength of an agent relative to other agents by having it play many games against those agents.

4.1 Scoring

It will be helpful to assign scores that represent how well an agent performed in a game. Let us say agents A_1, A_2 and A_3 compete in a game. To assign a score, we will look at (unordered) pairs of agents. Let us consider a pair of agents A_i and A_j (with $i \neq j$). If A_i finishes before A_j, then A_i gets 1 point and A_0 gets 0 points. Correspondingly, if A_i finishes after A_j, then A_i gets 0 points and A_j gets 1 point. In the (unlikely) case that they finish in the same move, both get 0.5 points.

Assigning points to pairs of agents allows us to create ELO ratings [ELO]. However, we can also simply add up the points to get the total points of an agent in a particular game. For example, if the agents finish in the order A_3, A_1, A_2, then A_3 would get 2 points, A_1 would get 1 point and A_2 would get 0 points.

4.2 Two Tournaments

Your agent can participate in two different tournaments. In **tournament 1**, your agent will compete against our agents. Your agent should get an average score of at least 1.8 points from 50 consecutive games. **Tournament 2** is just for fun and participation is optional. There your agent can compete against some of our stronger agents as well as the agents of other teams. Agents competing in tournament 2 will get a rating based on the ELO system [ELO]. Note that tournament 2 is very experimental and we are not sure how stable and meaningful the ratings will be. A key challenge is that a game might last many days if not all the agents are online at the same time, which means that their ratings (and potentially playing strength) can change significantly within a game.

4.3 Credentials and Client Implementations

We will commit one json file to your team repository for each tournament, which will contain the fields:

- `name` (a name for the agent),
- `pwd` (a password for the agent),
- `tournament` (the name of the tournament), and
- `url` (the server URL).

The assignment repository [GlAdv] 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 Section 4.4) 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.

4.4 The Server Protocol

This section 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 the moves you want to play, and the server responds either with an error message or with a list of positions that you should provide moves for in the next request. For the first request, you will have to send an empty list of moves to get the first positions from the server that require a move (see Figure 4 for an example).

```
// First request:
{'name': 'MyAgent', 'pwd': 'r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE',
    'moves': []}

// First response
{'errors': [], 'messages': [], 'positions': [
    {'id': '40#1', 'position': '1,-4;2,-4;3,-4;2,-5;3,-5;3,-6:3,1;...'},
    {'id': '7#3', 'position': '1,-4;2,-4;3,-4;2,-5;3,-6;1,-3:3,1;2...'}]}

// Second request
{'name': 'MyAgent', 'pwd': 'r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE',
    'moves': [{'id': '40#1', 'move': '2,-5;2,-3'},
              {'id': '7#3', 'move': '1,-4;1,-2'}]}
```

Figure 4: Example interaction with the server.

Please try to avoid flooding the server with requests. In particular, the server may return an empty list of positions if it is not your turn in any game. In that case, please wait for one second before sending another request.

The request You should send a `PUT` request to `[SERVER NAME]/play/[TOURNAMENT NAME]`, where `[SERVER NAME]` and `[TOURNAMENT NAME]` are provided by the configuration file (Section 4.3). The request body should contain a json object with the fields:

- `name`: the agent's name (from the configuration file).
- `pwd`: the agent's password (from the configuration file).
- `moves`: the moves the agent wants to do as a list. Each move is represented as an object with two fields: `id` is an identifier of the position (provided in the server responses) and `move` is the move the agent wants to do as described in Section 3.3.

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

- `positions`: a list of positions that you should send moves for in the next request. Each position is an object with two fields: `id` is an identifier so that your move can be linked to the position and `position` is the position/game state as described in Section 3.2.
- `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.

5 Submission

At the deadline, we will download a snapshot of your repository. It should contain

1. all your code,
2. a readme file explaining how to run your code (in particular how to run it with an arbitrary configuration file),

Furthermore, we will check if your agent has achieved the required performance in tournament 1.

6 Random Tips

- Make sure you have a good representation for FAUhalma positions.
- Make sure you get all legal moves (and only legal moves) for a FAUhalma position (maybe implement unit tests).
- Consider implementing and testing very simple agents before moving to more sophisticated algorithms.
- You can use the server to see how well your agent plays 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.

References

[CC] *Chinese checkers*. URL: `https://en.wikipedia.org/wiki/Chinese_checkers` (visited on 11/10/2021).

[ELO] *Elo rating system*. URL: https://en.wikipedia.org/wiki/Elo_rating_system (visited on 11/10/2021).

[GlAdv] *Assignment repository for adversarial search*. URL: https://gitlab.rrze.fau.de/wrv/AISysProj/ws2122/adv-search/assignment.