NAVIGATION
FUP
Homework assignments
Haskell Assignment #1
Haskell Assignment #2
Lectures
Labs
ALL COURSES
Summer 2017 / 2018
Older

Haskell Assignment #2

AI Sedma Player

Your task in this assignment is to program a strategy for a player of Sedma. Your strategy shall be represented by a function which will be called at every round. The function should select a card to play from player's hand based on the cards which are on the current trick. Additionally, your player is allowed to maintain its inner state, where you can remember any information you like (current cards in hand, previously played cards, current score, player name, ...). In order to maintain the state, you have to provide an update function which shall be called at the end of every round. Your strategy will be used for the two players in your team (A+C or B+D) and it will be tested against other strategies. A generic Sedma engine, capable of dealing the cards and simulating the game, is provided below.

Representation

The state of a player can be an arbitrary type you design. You **must**, however, make it an instance of the following type class.

```
data Player = A | B | C | D deriving (Eq, Show)

type Trick = [Card]

class PlayerState s where
   initState   :: Player -> Hand -> s
   updateState :: Trick -> Player -> Card -> Maybe Card -> s -> s
```

Function `initState` will be called at the beginning of the game with the player name (A,B,C, or D) and with the initial cards in hand (always 4 cards). It should create an initial state. Function `updateState` will be called at the end of each round with the following arguments:

1. the current trick (an ordered list of 4 cards)
2. the leader of the current trick (the one who played the first card in this trick)
3. the card played by this player in this trick
4. the new card obtained at the end of this round (if any)
5. current player state.

The function should return an updated state.

A player is represented by a function of the following type `AIPlayer`

```
type AIPlayer s = Trick -> s -> Card
```

It takes the following arguments: (1) the current incomplete trick (from 0 to 3 cards), and (2) current player's state. It must return one of the cards currently in the hand. As the cards currently in hand are not given to your AI player, you need to remember them in your state and maintain this information via `updateState`.

Call your function `player`.

```
player :: AIPlayer YourState    {- YourState implementing PlayerState -}
```

An example of a *cheating* player which always plays Heart 7 can be written as follows:

```
cheater :: AIPlayer s
cheater _ _ = Card Heart R7
```

This player is of course not a valid player and will be detected by the game engine.

Implementation

The following modules are provided and must be used unchanged.

- `SedmaBase.hs` contains Sedma basic types (cards, players, ..), some common functions you might use, and the above type class.
- `SedmaReplay.hs` is basically the solution of the previous Haskell assignment used to evaluate the winner.
- `SedmaGamble.hs` contains the game engine that you can use to test your player.

The most important function in `SedmaGamble.hs` is

```
gamble :: (PlayerState s, PlayerState t) =>
          AIPlayer s -> AIPlayer t -> Cards -> Cards
```

which takes two AI player strategies used by teams A+C and B+D. Additionally it takes an ordered (shuffled) deck of 32 cards. It simulates the game with the four players on this deck and returns the list of cards as they were played. This list can be then evaluated by function `replay` from the previous assignment.

Testing

For testing, you should develop a `silly` player, which always plays the first card from his hand. The state for this player needs only to remember the current cards in hand. Then you should develop a better player, for example, based on a simple decision tree (or a set of rules). Your player should be able to beat `silly` by at least *n/2* points in *n* consecutive games with random (shuffled) decks. A player with such a performance shall be enough to reach 6 points from this assignment. For more points, you might need a better performing player.

You can download the above Haskell modules and a set of 20 testing decks in module `SedmaDecks.hs`. Do not submit the downloaded modules. Submit just the file implementing the function `player` and your state.

Evaluation system

You will be able to upload your solution to the evaluation system soon. However, the evaluation scripts will be most likely running after the weekend, when they are finished. We are sorry for the inconvenience.