XForms Blackjack documentation

Daniel Meint <d.meint@tum.de>

Felix Hennerkes <ga38hom@mytum.de>

Janik Schnellbach <janik.schnellbach@tum.de>

Maximilian Karpfinger <maximilian.karpfinger@tum.de>

Table of Contents

The component model

Modeling the model

In a MVC-architecture the model has the control of the application's logic. Furthermore it is resposibel to store data in a database. In order to maintain data, several objects with unique functionality are needed. All these collaborating objects provide different attributes and methods. In the following the data model will be described at first. Thereon the design decisions will be discussed. The second part consists of the implementation of the game, followed by a discussion on implementation details and what problems were faced.

The database consists of a list of games. For that a class `game` is needed which stores every information about the instance of a blackjack game. A game object has multiple attributes and functions.

- `id` to distinguish games
- `state` stores the state of the game

Regarding `game` functions, their main purpose is constructing,initializing and resetting game objects. Because every game object also stores all the attending players and a dealer, the functions essentially execute operations on all players. In addition to the basic functions, `game` provides the function `evaluate`. However the logic of evaluating a player is sourced to the `player` class. Another object stored in a game object is the chat.

The `player` has the following attributes.

- `name` stores name of the player
- `state` stores the state of the player
- `insurance` stores wether or not a player chose to take insurance
- `balance` stores the total balance of a player
- `bet` stores the player's bet in a round
- `profit` stores the player's profit in a round

Another object stored inside the player is the players hand, which holds the cards. As to the `player` functions, contiguous to the constructors and setters, there are alot of functions that contain the game's logic. Depending on the game's state, a player can take several actions, which are passed over to the `player` class by the API. Following functions are supplied.

- `hit` draws another card for the player
- `stand` the player doesn't want more cards
- `double` doubles the player's bet and draws one last card
- `insurance` the player takes insurance
- `evaluate` determines the player's profit

Alongside the listed functions there are some helpers, which provide logic for finding the next player and joining or leaving a game.

Since the players all play against the dealer, another class `dealer` is required. It has no attributes, but it stores the deck of the game and likewise the players a hand object. Concerning the functions of the class, `evaluateInsurance` is a helper for evaluating a players profit. The function `deal` gives every player two cards and also draws for the dealer.

The `hand` class has the attribute `value` which stores the value of the contained cards. On one hand this class provides helper functions for evaluating a player's hand. On the other hand it supplies the `getOptimalSum` function that calculates the optimal sum of a hand. This is needed because the card Ace can have the value 1 and 11.

The `card` class consists of two attributes.

- `value` stores the value of a card
- `suit` stores the suit of a card

Design decisions and discussion

why we chose to have object hand

why we have usr and player

Implementation

deck is real not generating random cards

optimal sum recursive

Implementation decisions and discussion

dealer drawing at first

profit for front end

how to deal with broke players

problem of trying to replace node multiple time -> work around

double and bust call next() -> problem of evaluate data not in db if it was last player