Assignment 9: 2-Player Client-Server Yahtzee

CS 5010
Released: November 27, 2017
Due: December 4, 2017

Abstract

Yahtzee is a dice-rolling game. The game requires 5 dice and a scorecard, and can be played by (theoretically) any number of players. Gameplay consists of two phases: the Rolling phase, and the Scoring phase. In the Rolling phase, players can roll as many of the dice as they want, for up to 3 times total. At each roll, players can keep or re-roll as many dice as they want. In the Scoring phase, the player chooses which entry on the scorecard they want to fill, given the final values on the rolled dice. The player with the highest number of points on the scorecard at the end wins. As you can imagine, it's important to be familiar with the scorecard and figure out a strategy to maximize points. However, in this assignment, you're less concerned with winning, and should be more concerned with how to make the game play fun and easy for your players.

1 Assignment Objectives

The objectives of this assignment are to:

- Write a client to a given protocol
- Build a text-based UI that runs on the command line
- Implement a client-server application

2 Background

We learned that a client server application requires a server application, a client application, and a protocol defined to ensure consistent communication between the two. In this assignment, you are going to write a client app that connects to an already-written server application, implementing the game protocol.

In the end, you will have a game that you can play!

3 Introduction to Yahtzee

Yahtzee is a dice game. The goal is to fill out a score card, based on rolls of the dice. There are 12 rounds to the game– one for each line on the scorecard. Each player goes through a round of rolling the dice, and then chooses which row of the scorecard to fill out. This results in points.

3.1 The Scorecard

Below is the scorecard used for Yahtzee. Note, you do not need to implement this—scoring is done on the server. But being familiar with the scorecard will be helpful for your implementation. The ID column indicates what the score row is referred to in this game implementation.

Upper Section

Pattern		Condition	Score	ID
Aces	⚀ ⚀ ⚀ ⚀ ⚀	none	total value of all aces (ones)	Aces
Twos	⚁ ⚁ ⚁ ⚁ ⚀	none	total value of all twos	Twos
Threes	⚂ ⚂ ⚂ ⚂ ⚀	none	total value of all threes	Threes
Fours	⚃ ⚃ ⚃ ⚁ ⚀	none	total value of all fours	Fours
Fives	⚁ ⚄ ⚁ ⚄ ⚄	none	total value of all fives	Fives
Sixes	⚅ ⚅ ⚅ ⚁ ⚀	none	total value of all sixes	Sixes

Lower Section

Pattern		Condition	Score	ID
Three of a kind	⚃ ⚃ ⚃ ⚁ ⚀	3 or more equals	total value of all dice	ThreeOfKind
Four of a kind	⚄ ⚄ ⚄ ⚄ ⚂	4 or more equals	total value of all dice	FourOfKind
Full House	⚂ ⚂ ⚂ ⚁ ⚁	3 equals + 2 other equals	25	FH
Small Straight	⚀ ⚁ ⚂ ⚃ ⚀	sequence of 4 or more	30	SS
Large Straight	⚁ ⚂ ⚃ ⚄ ⚅	sequence of 5	40	LS
Yahtzee	⚃ ⚃ ⚃ ⚃ ⚃	5	equals 50	Yahtzee
Chance	⚁ ⚅ ⚃ ⚁ ⚀	none	total value of all dice	Chance

4 How to build this?

Notice, the instructions so far have been all about implementing the client. You will be provided with a server implementation to use for development.

4.1 How to run the server locally

You've been given the .jar file that runs the server. To run it locally, save the jar file somewhere. Use the terminal to navigate to the jar file. Run like:

```
java -jar YahtzeeServer.jar 1200
```

(the 1200 is the port number– it can be anything over 1024, mostly). The server waits for 2 clients to connect, then starts the game. Make sure your client connects to localhost and on the specified port.

Yahtzee is a long game. To facilitate development and debugging, you can start the server in DEVELOP mode, which will run the game with only 1 player and 3 rounds.

```
java -jar YahtzeeServer.jar 1200 DEV
```

If you're having trouble, you may need to make sure your java is pointing at a Java 1.8 VM to run it.

After you start the server, you can run your client.

To run your client from IntelliJ while you're developing, you can modify the Run configuration to take the appropriate command line argument by going to "Run → Edit Configurations... ". Choose your configuration on the left, and put "localhost <portnumber>" in the Program Arguments field.

4.2 Requirements

Your client must:

- Run from the command line, given a hostname and port number
- Connect to the server via a socket
- Get info from the user and format it appropriately to send to the server.

5 What To Turn In

When submitting your assignment, you should continue using the same Maven archetype that we used in previous assignments.

You will want to submit the following:

- Class PlayYahtzee. This class will be assumed to be the starting point of your program, and will be called from the command line.
- All classes that you have developed for this assignment
- All classes that you developed to test your code
- A UML diagram showing the design of your program
- A brief write-up, which summarizes the main relations between your classes, and how does your program handle errors and/or exceptions.

6 The game protocol

A protocol is defined by a set of “Frames” or messages that are sent between the server and client. The frames specify the beginning and end of a message, and the actual message is in the “payload” of the frame. Consider that our clients and servers are listening to streams of data coming in. How does your client know where the end of the frame is, in order to parse the message?

In this protocol, the newline (\n) character is the end of a frame. The beginning of a frame is a tag (string) followed by a colon (‘:’), which is then followed by a payload. Your client recognizes receipt of a frame, parses the frame type from the frame header, and then uses that information to parse the payload and respond accordingly.

In the next sections, the client and server frames are defined. After that, the list of the interaction of the frames.

6.1 Client Frames

These are the messages that can be sent from the client to the server.

1. KEEP_DICE
   Payload: 10 ints. The first 5 are the original dice that were passed in “ChooseDice”, and another 5, which must be 0 or 1s to indicate whether to keep the relevant dice. If the original dice sent were “1 2 6 4 1” and the player wants to keep the 1st, 2nd and 5th, the payload would be “1 2 5 4 1 1 1 0 0 1”.
2. SCORE_CHOICE
   Payload: Name of score to take
3. ACK
   Sends an acknowledgement of a message. Payload: an optional string describing the acknowledgement.
4. PRINT_GAME_STATE
   Prints the current game state on the server terminal. Payload: empty.

6.2 Server Frames

1. START_GAME
   Payload: empty
2. START_ROUND
   Payload: Which round
3. START_TURN
   Payload: empty
4. CHOOSE_DICE
   Payload: 5 ints, with values 1-6, each representing a given die roll.
5. INVALID_DICE_CHOICE
   Payload: None
6. CHOOSE_SCORE
   Payload: 5 ints that are the current dice rolls, and then a sequence of strings of the names of the score slots to be chosen from.
7. SCORE_CHOICE_INVALID
   Payload: the same payload as CHOOSE_SCORE.

8. SCORE_CHOICE_VALID
   Payload: A list of all the scores on the scorecard and their values. Value is -1 if it has not been used yet. Order is arbitrary. Total is at the end.
   Example: Aces -1 Twos 6 Fives 5 3K 6 [...] Total 75
9. TURN_OVER
   Payload: empty
10. ROUND_OVER
    Payload: which round is over.
11. GAME_OVER
    Payload: A list of the final scores for each player.
    Example: Player1 109 Player2 74
12. ACK
    Payload: an optional string describing an acknowledgement of receipt of a message from the client.
13. INFO
    Payload: a string providing an informational message from the server to the client.

6.3 Interaction

1. Server starts
2. First client/player connects
3. Second client/player connects
4. Server sends "StartGame" to all clients
5. Game play rounds begins. Server sends StartRound message to all players.
   For each client/player, this is the sequence for each player in each round:
   (a) Server sends StartTurn message
   (b) Start the rolling phase. While the player has rolled < 3 times or not kept all the dice:
      i. Server rolls the dice and sends ChooseDice message
      ii. Client sends KeepDice message
      iii. Server sends valid/invalid KeepDice choice message
   (c) Start the scoring phase.
      i. Server sends set of score choices for client to choose from
      ii. Client sends score choice
      iii. Server sends valid/invalid score choice.
   (d) Server sends TurnOver message.
6. repeat for each player
7. repeat for each round (13 rounds total, unless in DEV mode)
8. Server sends GameOver message