Introduction to Java
CS9053
Tuesday 6 PM – 8:30 PM
Prof. Dean Christakos
Final Project
Assigned: Nov. 24, 2020
Due: Dec 14, 2020

Goal

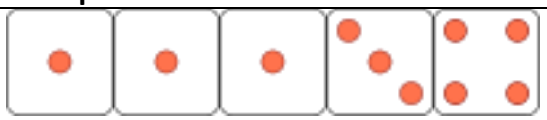
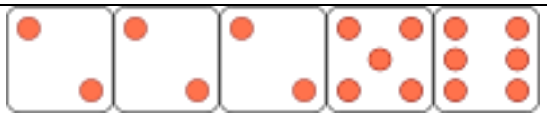
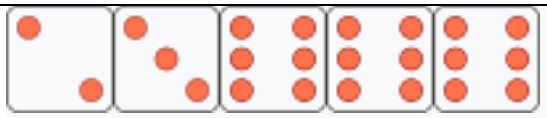
You are going to create the game Yahtzee in Java. This is a relatively simple game to learn and involves 5 dice and 1 to 4 players. In a round, each player takes a turn. On each turn, a player rolls the dice with to get them to one of the 13 categories listed. If the first roll doesn't fulfill one of the categories, the player may choose to roll any or all of the dice again. If the second roll fails, the player may roll any or all of the dice again. By the end of the third roll, however, the player must assign the final dice configuration to one of the thirteen categories on the scorecard. If the dice configuration meets the criteria for that category, the player receives the appropriate score for that category; otherwise the score for that category is 0. Since there are thirteen categories and each category is used exactly once, a game consists of thirteen rounds. After the thirteenth round, all players will have received scores for all categories. The player with the total highest score is declared the winner.

Scoring

The scoring is described here, based on the Yahtzee rules described in Wikipedia: https://en.wikipedia.org/wiki/Yahtzee

We divide the sections into **Upper** and **Lower Sections**.

The **Upper Section** can have any combination of dice and the score relies on the face of the dice.

Categories	Description	Scores	Examples
Aces	Any Combination	The sum of the dice with the number 1	 Score: 3
Twos	Any Combination	The sum of the dice with the number 2	 Score: 6
Threes	Any Combination	The sum of the dice with the number 3	 Score: 12
Fours	Any Combination	The sum of the dice with the number 4	 Score: 8
Fives	Any Combination	The sum of the dice with the number 5	 Score: 5
Sixes	Any Combination	The sum of the dice with the number 6	 Score: 18

If a player scores a total of 63 or more points in these six boxes, a bonus of 35 is added to the upper section score. Although 63 points corresponds to scoring exactly three-of-a-kind for each of the six boxes, a common way to get the bonus is by scoring four-of-a-kind for some numbers so that fewer of other numbers are needed. A player can earn the bonus even if they score a "0" in an upper section box.

The **Lower Section** requires the five dice to fit a specific pattern for each category

Categories	Description	Scores	Examples
Three of a Kind	At least three dice of the same	Sum of all dice	 Score: 17
Four of a Kind	At least four dice of the same	Sum of all dice	 Score: 24
Full House	Three of one number and two of another	25	 Score: 25
Small Straight	Four sequential dice (1-2-3-4, 2-3-4-5, or 3-4-5-6)	30	 Score: 30
Large Straight	Five sequential dice (1-2-3-4-5, 2-3-4-5-6)	40	 Score: 40
Yahtzee	All five dice the same	50	 Score: 50
Chance	Any combination	Sum of all dice	 Score: 13

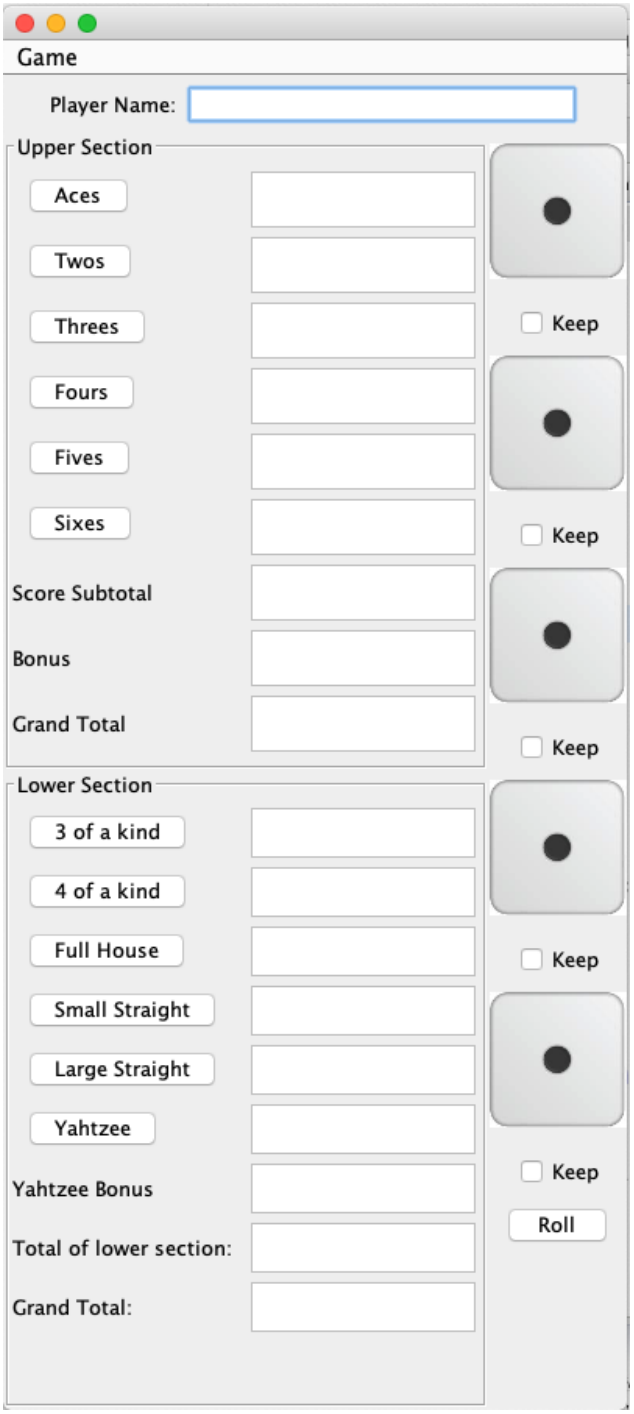
The astute observer will notice that a Yahtzee is **also** a “Full House”, or can fulfill a “chance” box, and that all of the Lower Section categories could easily be fit into any of the upper section categories. The official rules require that the categories be filled in the following way:

- If the corresponding Upper Section box is unused then that category must be used.
- If the corresponding Upper Section box has been used already, a Lower Section box must be used. The Yahtzee acts as a Joker so that the Full House, Small Straight and Large Straight categories can be used to score 25, 30 or 40 (respectively).
- If the corresponding Upper Section box and all Lower Section boxes have been used, an unused Upper Section box must be used, scoring 0.

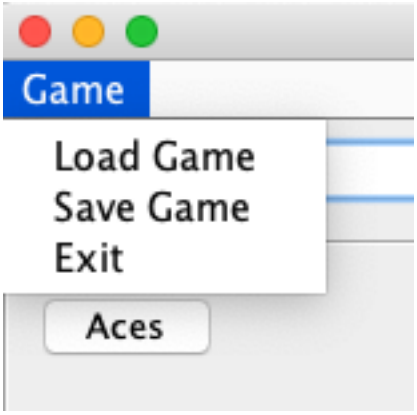
The game

This is probably the best place to start. To keep things simple, this will be a one player game. You'll have to create the UI and the internal logic that keeps track of which round/turn a player is on and how many rolls they have on his turn.

Here's a model for what your game design could look like:

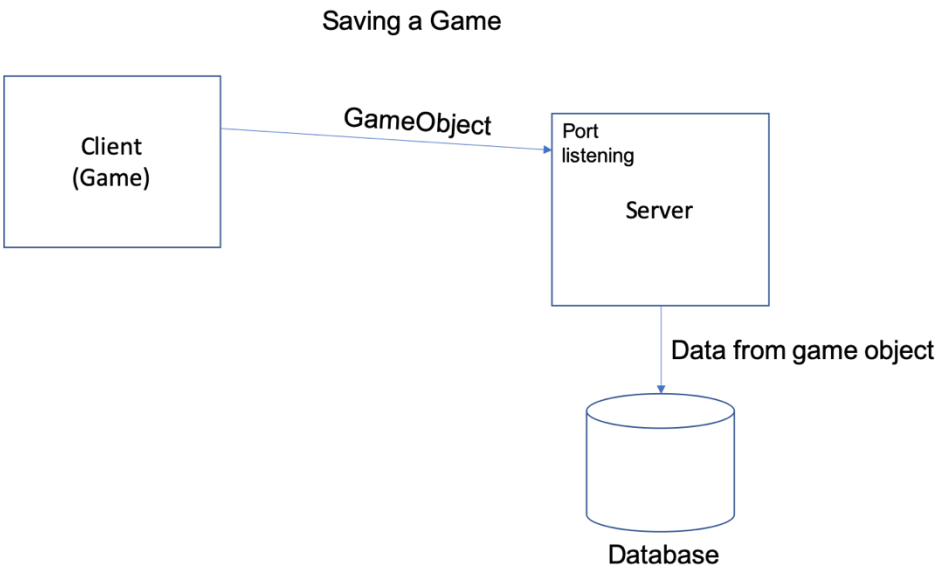


Saving and Loading Games

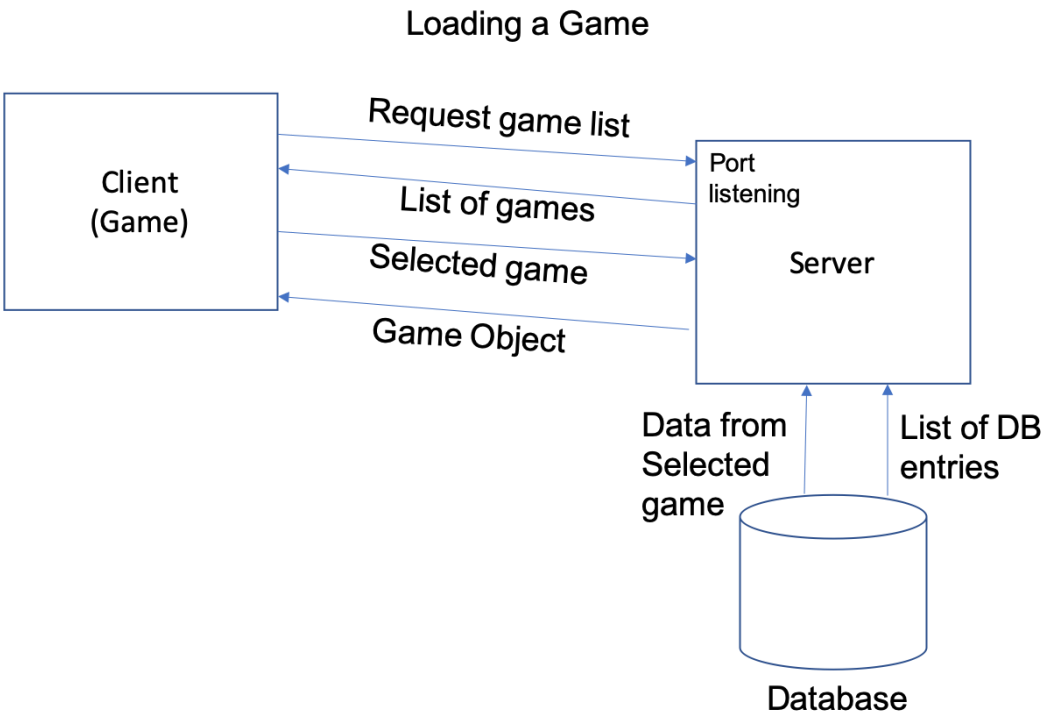


The next feature is a “save game” feature. From the file menu, the user should be able to save the progress of the game. The trick is that the games should be saved in a database, and that database should be located on a remote server.

To save the game, the game data and state should be marshalled in to an Object, and the object should be sent to the remove server. The server should receive the object and store the data in a database table.



To retrieve a saved game, the client should send a request to the server. The server should send a list of games indexed by player name and time saved. When the user selects a game, it sends the request to a server, the server receives the requests, gets the data from a database, marshalls it into an object, and sends the object back to the client.



The client then receives the object and adds its data to the game, and the player picks up where he left off.

Implementation

You are going to have a fair amount of flexibility in implementing this.