Clicker 2

Due: Wednesday, April 24 @ 11:59pm

Updates and Pitfalls

-

Objectives

1. Add networking and concurrency to clicker 1
2. Practice reading code and using existing code to accomplish your goals

Description

In clicker 1 we made a simple clicker game that ran as a desktop app with a manual save feature. For part 2 we will convert this into a web app that can support multiple users and has an autosave feature. Users will go to the website and enter their username to start playing the game. When they return to the site later and enter the same username they will return to all their progress including the idle gold collected while they were away. This idle income and their purchased buildings will persist even after the server is restarted.

The structure of this app will follow the structure covered in lecture (without the desktop version). Keep this in mind while reading through the provided code.

We will focus on the networking involved to accomplish this goal and have users only enter their username to play, though you can add the authentication from CSE115 to add a proper login system.

Project Structure

1. Create a new project in IntelliJ
2. Pull the Scala Examples repo and copy the clicker2 package into the src folder in your new project
3. For your own testing, also copy the clicker2webserver module into repo. This module is not graded

Testing Objectives (0 points)

There are no testing objectives for this assignment.

Objectives (50 points)

A major goal of this assignment is to practice reading existing code and using the provided code to complete these objectives.

Objective 1 (20 points)

Complete the GameActor class.

Since our app can have many simultaneous players we need a way for the server to track all players concurrently. GameActors will be used to track each player's game. The constructor of this class will take the username of a player and it will create and run the game for that player. This Actor will respond to the following message:

- Setup: Initialized the game. If this is a new player, create a database entry for this player. If the player has a game saved in the database, load that game
- Update: Update the game using the current time in nanoseconds, then respond to the sender with a GameState message containing all the data from this game as a JSON string (Call the toJSON method from clicker 1 to get the game state)
- Save: Save the game in the database
- ClickGold: The player for this Actor has clicked the gold button
- BuyEquipment(equipmentID): The player for this Actor is attempting to buy equipmentID

Each time you interact with the database, call the appropriate method(s) from the Database object. Do not write your own database methods or modify the provided methods.

Objective 2 (15 points)

Setup the CickerServer class as a TCP socket server.

The ClickerServer class will accept TCP socket connections from the Python web server. This is the class that will receive message from Python as JSON Strings and process the requests. To setup this class:

- Listen on port 8000 for socket connections. You may assume that only 1 client (web server) will be connected at a time (Note that our architecture is setup to scale to multiple web servers)
- React to GameState messages by sending the state to the web server followed by the delimiter (The web server will read from the socket as a single stream of data. The delimiter allows to know when one game state stops and when the next begins)

Objective 3 (15 points)

Complete the CickerServer class

Now that the connection to the web server is established we can react to messages from this server. This server will send JSON strings in the format

{“username”:”user1234”, “action”: <action_type>}

The action_type will be one of the following:

- “connected”: The user has connected. Create and set up a GameActor for them. Remember to store their ActorRef and remember their username
- “disconnected”: The user has disconnected. Shut down and remove everything associated with this user (Their game is still saved in the database for when they reconnect). You can stop an Actor by sending it the PoisonPill message
- “clickGold”: The user clicked the gold button
- “buyEquipment”: The user clicked a buy equipment button. This JSON string will contain a 3rd key-value pair with the key “equipmentID” with a value of the equipment type that was clicked

To finish this class, also react to the following messages:

- UpdateGames: Send the Update message to all GameActors
- AutoSave: Send the Save message to all GameActors

Bonus Objective (25 points)

Add Alchemy to the game and a new type of equipment.

You do not have to add Alchemy to the database or the front end (you should for your own version of the game, but it won’t be graded).

Alchemy specs:

- Equipment id == “alchemy”
- Name == “Alchemy”
- 40 gps
- 5000 gpc
- 10000 initial price
- 10x price after each purchase