# Exercise7EscapeRoomAdmission

Jump to bottom

sethnielson edited this page 6 days ago · 2 revisions

# Exercise 7: Escape Room with Payment

| | |
|---|---|
| Assigned | 9/25/2019 |
| Due | 9/30/2019 |
| Points | 25 |

## Overview

This assignment should be fairly simple. Most of the hard parts of this code are already written. For this lab, you are going to create an Escape Room that requires a payment to play.

## Using the Bank

By now, you should have a bank account. If you don't please message the TA.

In the Bank source code directory is sample code for accessing the bank from a program. You do not have to write the packets yourself. Simply use the `BankClient` object that is already provided for your use. If you don't have this code, or don't know how to use it, message the TA.

# The PRFC and the Protocol:

You should, *as a team* have already been creating a standard for this in class. Your implementation should follow your team's standard *meaning your protocol should be interoperable with your team*.

Although you can define the PRFC yourself, and the implementation that goes along with it, I have to write an auto-grader that can work with it as well, so you have to write a library that I can import and use. I won't tell you the names of the classes of your packets because I don't even know *how many* packet types you will define. But you must define functions that allow me to build the right types of packets, etc. Let's first define the game protocol, and then I'll explain the auto-grade protocol.

For the game:

1. The game client sends a packet indicating a username and a desire to play.
2. The game server sends back a packet with a unique id, and an amount to pay to play. I won't pay more than 10 bitpoints to play, so be careful what you ask for.
3. The game client gets a receipt and receipt signature indicating payment with the memo field set to the unique id. It sends this receipt/signature to the server.
4. The game server, if the amount is correct and the id is correct, now permits the user to play.
5. If the payment is correct, the game proceeds normally with game commands and responses until the game is over. The server should close the connection (without sessions, it just shuts down its own protocol).
6. IF THE PAYMENT IS WRONG, the game sends back a response packet with the response set to empty and the status set to `"dead"`

To make this work with the autograder, please do the following:

1. Use the same autograder packets as for the previous lab. The first packet is the `AutogradeStartTest`
2. Just like the previous lab, you will transmit your "packet" file in the `AutogradeStartTest`. However, this packet file will be a little bit different; please see the specifications below.
3. Once you receive the `AutogradeTestStatus` indicating that submission was successful, you need to send your packet for requesting to start a game.
4. You will receive a packet with info about paying the bank.
5. Once the bank is paid, send the receipt and receipt signature to the autograder. This will start the protocol described above with the autograder as the game server. You will be tested on both correct and incorrect bank responses. Each new packet indicating a desire to play should start the protocol above.
6. Once the client has escaped, you will receive the `AutogradeTestStatus` with indicating that the server side was also successful.

The the packet file, I don't know how many packet types you have or what their names are. So, please use the following factory functions defined within your packet file.

1. FIRST:, a function `create_game_init_packet(username)` that creates a packet for initiating the game. If your users must login, please create a test username that does not require it, or can have it hardcoded.
2. SECOND:, a function `process_game_init(pkt)` that returns the username
3. THIRD:, a function `create_game_require_pay_packet(unique_id, account, amount)` that will produce a packet that requests a particular amount from the bank
4. FOURTH:, a function `process_game_require_pay_packet(pkt)` that will return the unique id, the deposit account, and the amount.
5. FIFTH:, a function `create_game_pay_packet(receipt, receipt_signature)` that will produce a packet proving payment.
6. SIXTH: a function `process_game_pay_packet(pkt)` that will return the receipt and receipt signature from the packet.
7. SEVENTH: a function `create_game_response(response, status)` that will produce game responses with the textual response and the status
8. EIGHTH: a function `process_game_response(pkt)` that will return the game response and the status
9. NINTH: a function `create_game_command(command)` that will produce a game command
10. TENTH: a function `process_game_command(pkt)` that will return the game command
11. As before, there will be `AutogradeTestStatus` packets returned after the submission begins, the client is finished, and the server is finished.

## NOTES

1. It doesn't matter if you create one packet type or two dozen so long as your functions create the correct type.
2. These functions are not methods. Do not put them in the packet class. These should be standalone functions in the module
3. The process functions should make sure that the "type" (however defined) is correct. For example, if your `process_game_command` packet gets a game pay packet, it should *throw an exception*.
4. Although this seems a little tedious, you should be adapting most of your code from the previous lab. There should be very little new code to write.

## Autograde Stuff

As usual: 20194.0.0.19000 and port 19007

▼ Pages 19

Find a Page...



### Clone this wiki locally

```
https://github.com/CrimsonVista/20194NetworkSecurity.wiki.git
```