Project: SNAKE?!

Description: This projects will implement the famous snake game with basic userbase and leaderboards functionality.

This project will have it’s code split between three modules as per the MVC architecture. The View and Controller modules will be on the client side while the Model module will be on the server side.

Game flow:

1. The user loads the application.
2. The user is asked to login/register (login details include server ip).
3. After login is authorized, The actual game window is loaded into the frame.
4. Game starts and runs until:
5. The snake dies.
6. The user closes the application.
7. The server closes.

5. Upon game end the score of the user will be compared with his previous high score and the new highscore will be saved in a database.

6. GAME OVER message will be popped to the user then the user gets a choice to play a new game or close the application.

The application has two states, Before and after the login screen.

Login screen:

One big gray panel with 3 text boxes , 3 labels and 2 buttons.

The user will need to input a username , password and server ip.

The two buttons will be login and/or register.

The login button will check connectivity with the server and send a query to the model about given username and password.

Upon receiving authorization the Controller will swap the View to the game screen.

The register button will check connectivity with the server and send a query to the model checking wheter given username already exists. If not – model will add user to the database via query and send confirmation of registry to the Controller which will demand the user sign-in.

Game screen:

This frame will show the user the game as it’s running.

The View module will paint a panel on the frame.

The View module will be listening for any Key events and support a specific list of keys to cause actions in the game.

For each iteration of the movement of the snake, The Controller will setup a connection with the model (the connection will be through a socket as the Controller module is client side while the Model module is server side).

Database:

In order to switch between the login screen and the game screen , The user needs to login , this process will be done through database query.

The database will be set up in the following fashion:

A table with the username as key and each key will have a password value , a high score value and level of authorization value (user and admin).

Users with authorization 1 (player) will not be able to interact with the database beyond indirect queries from registry , login , and saving score when game ends.

Users with authorization 2 ( admin) will be able to manually change authorization levels of other users and deleting the said user/player.

This project will adhere to the MVC architecture and the three modules (Model , View , Controller) will be set as following:

1. Model – The model will hold all communications with the database. The model will calculate the next turn upon receiving the game state from the Controller.
2. View – The View only paints the panel on the client side frame. The frame will consist of the game panel which will be 800x600. Above the game panel there will be 2 buttons. The first will be titled “HELP” which will cause a pop-up to show consisting of a text message explaining the game rules and listing all the inputs allowed. The second button will make the leaderboard pop up in a separate frame. The view will also have several action listeners that will relay the inputs to the controller.
3. Controller – The Controller will be listening for any events to happen in the View. Every set amount of time the Controller will send a request to the Model to calculate the next turn. Upon receiving the response the Controller will create the panel which will be sent to the view to be painted on the frame. The Controller also will manage the login attempt of the user on the client side. Upon button click of either login or register , the Controller will create the appropriate query to the Model and handle the response from the Model.