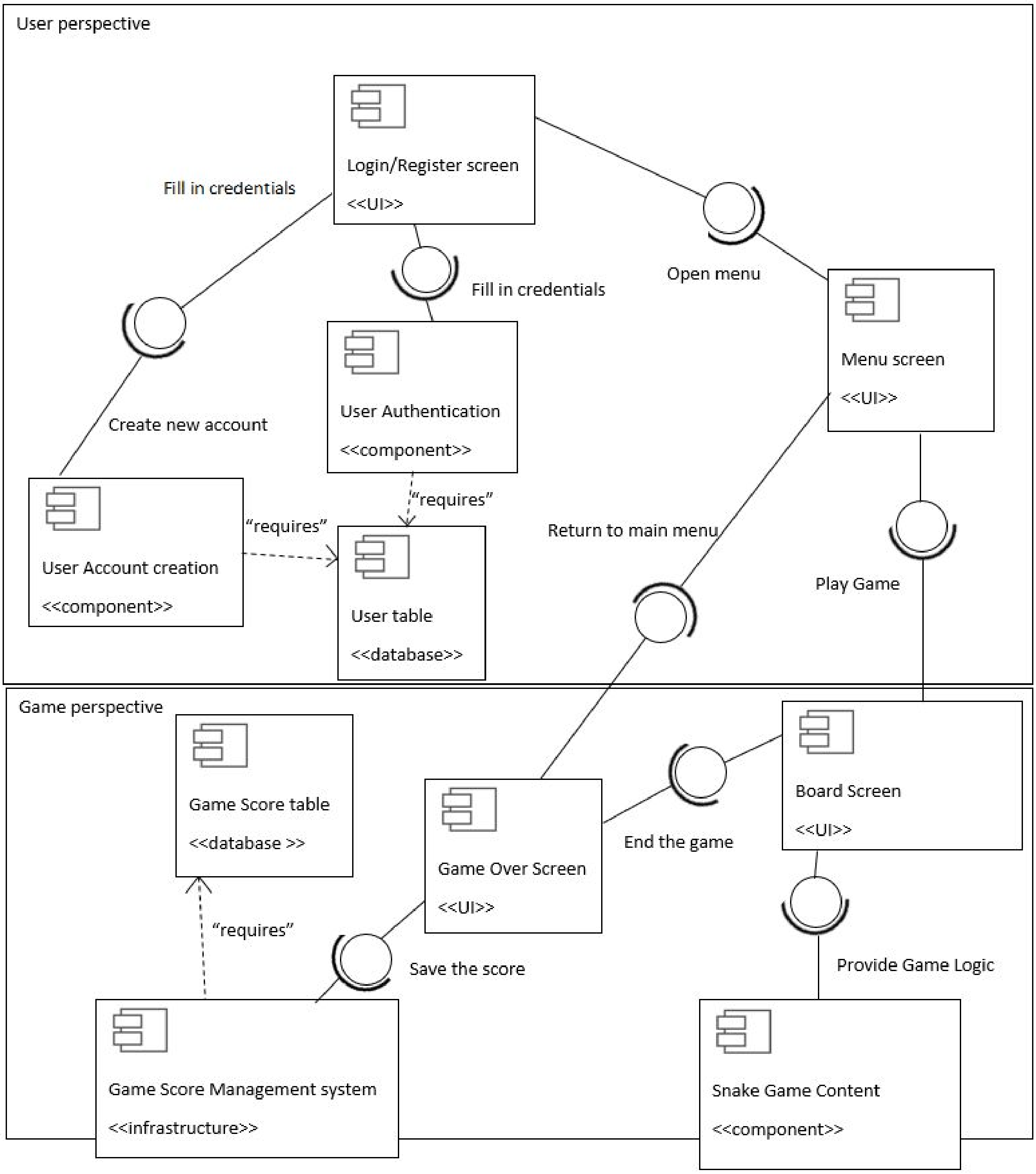
Exercise 2

Architecture diagram

diagram:



descriptions:

1. Login/register screen: the login/register screen provides the user with 2 options:
   a. To create a new account for the Snake Game.
   b. To login with an existing account.
2. User Authentication: When the user has already created an account, he/she can login by pressing the login button which appears on this screen. Then the user can fill in his/her login credentials with his/her user account, which is required for user authentication.
3. User Account creation: When the user doesn't have an account, he/she can press the register button which appears on the screen. Then the user can create a new account by filling in some username and password for the Snake Game.
4. User table: If the user chooses the login option, then the user authentication requires to retrieve the given username and password from the user table in the database. If the user chooses the register option, then the user account creation requires to insert the given username and password into the user table in the database.
5. Menu screen: When the user has successfully logged in/registered, he/she will be provided with menu options where he/she can choose:
   a. to play the game.
   b. to view the leaderboard.
   c. to change the color of the snake/food.
   d. to change the user password.
6. Board screen: When the user chooses to play the game, a board screen will be provided where the actual Snake Game takes place. In the board screen, the user has been provided with the option to pause the game in case he/she wants to take a break.
7. Snake Game Content: The user starts the game with a small and slow snake which can make moves in 4 directions (North, East, South, West). When the snake grows after a while by eating more and more food squares, the snake speed gets increased and the game score gets incremented. When the snake collides with one of the window edges or with its own tail, the snake dies.
8. Game Over Screen: When the snake dies, the game is over. The Game Over screen provides the user to view and save the game score he/she has achieved. Furthermore, it provides an option to return to the menu screen in case the user wants to play the Snake Game again.
9. Game Score Management System: When the user chooses the option to save his/her game score, he/she can also choose a Game name before saving the score. This Game name will be displayed with the game score on the leaderboard.
10. Game Score table: When the user saves his/her score, the given score and game name will be saved/inserted into the Game Score Table in the database for the current user.

motivations:

The architecture we've decided to use for our game is the layered architecture.
This is because our architecture diagram can be divided in 2 main layers:

1. The user perspective layer

2. The game perspective layer

The user perspective layer focuses more on the user-related structure parts where the user can have a view from his/her own perspective. The game perspective layer focuses more on the game itself where there can be made no user-related modifications. Features like user authentication, user registration and choosing menu options such as changing the password have nothing to do with the game itself, while features like playing the game, pausing the game and saving the game score into the database table can be considered as game-related features.

Given the structure and design of our application, there were other architectural pattern which were not applicable for use. For example, the reason we didn’t choose the model view controller architecture is because our architecture diagram isn’t layered in such a way that the user view (fxml files), the controller (login/register/menu/game-ended controller) and the model (database) are separated layers in our design. Another example which would not fit in our application is the pipe-and-filter pattern, since this would be more useful for applications which require data analysis and data transformation. If we decided to use the pipe-and-filter pattern, it could become a performance-killer of our simple-structured game and it could add too much overhead. Furthermore, we could have decided to use the ports-and-adapters pattern, but this would require to separate our main functionality from our external services. However, we have decided that our game doesn’t need external services, so this pattern also doesn’t really fit in our architecture design.

To sum up our motivation, for our layered architecture the game perspective layer provides services to the user perspective layer and the user perspective layer consumes those services from the game perspective layer.