Knowledge Siege: Simulation Design and Development

<Emirhan Efe Yaşar, 87050>
<Spring 2025

Part 1 – General Demo Information

Pre-Registered Player

A sample user has been pre-registered in the system:

- **Username:** Emirhan
- **Password:** 1234

The user logs in through the initial login screen. Upon successful authentication, they are directed to the main menu and then into the game screen.

During gameplay:

- Info ShotBoxes (💡) increase the score.
- Question ShotBoxes (❓) reduce health.

The game progresses through three levels. Reaching 150 points results in causing victorypanel opening.

Game log (log.txt) includes:

- **An information has hit the player.**
- **Score updated: 60**
- **Player is dead. Game Over.**

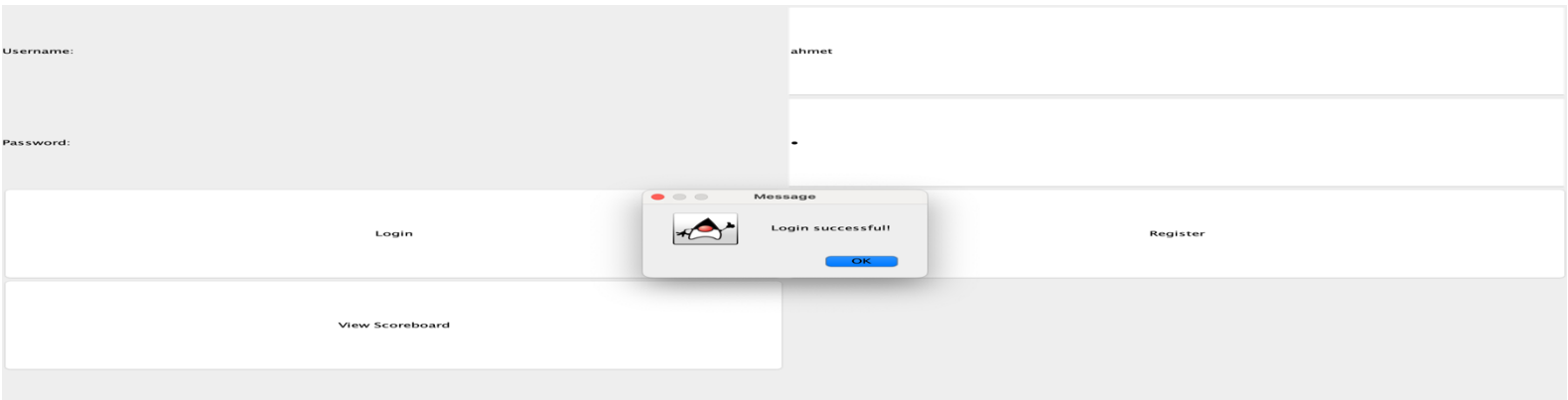
Final score is saved in scoreb.txt:

- Emirhan,150

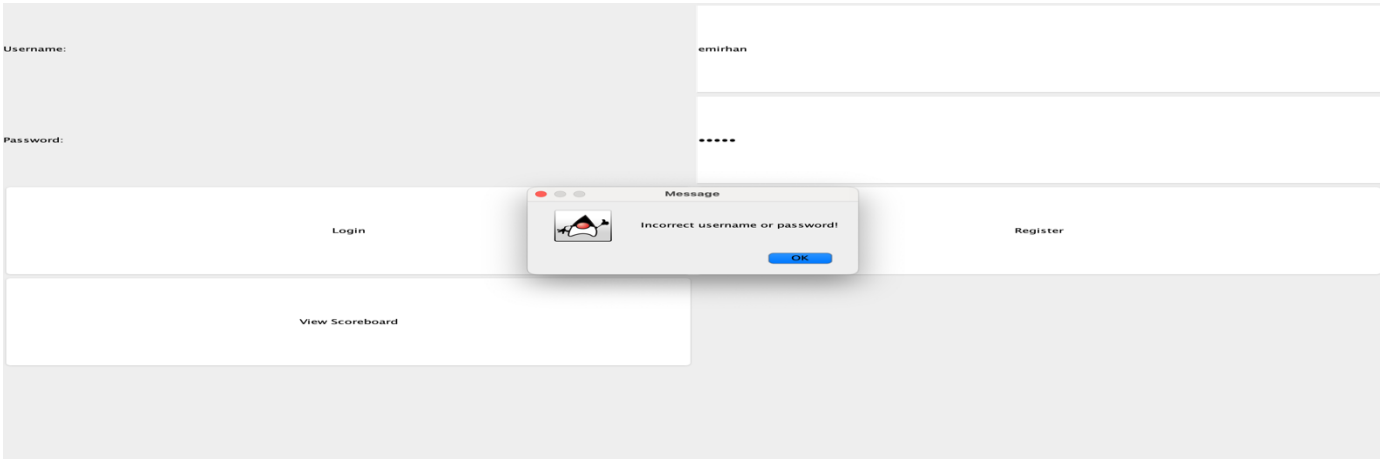
This score appears in the scoreboard panel, ranked among all players.

New Player

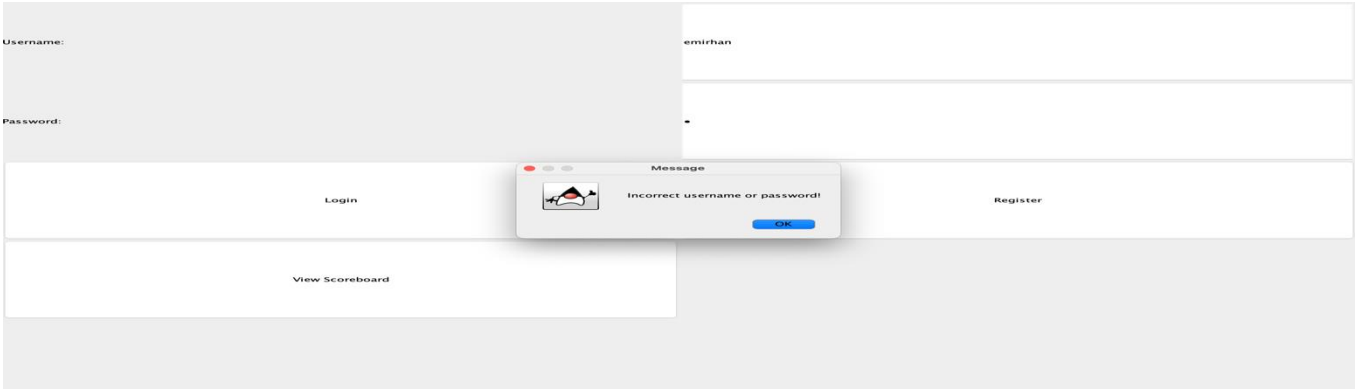
users can register through the login panel by entering a unique username and password..



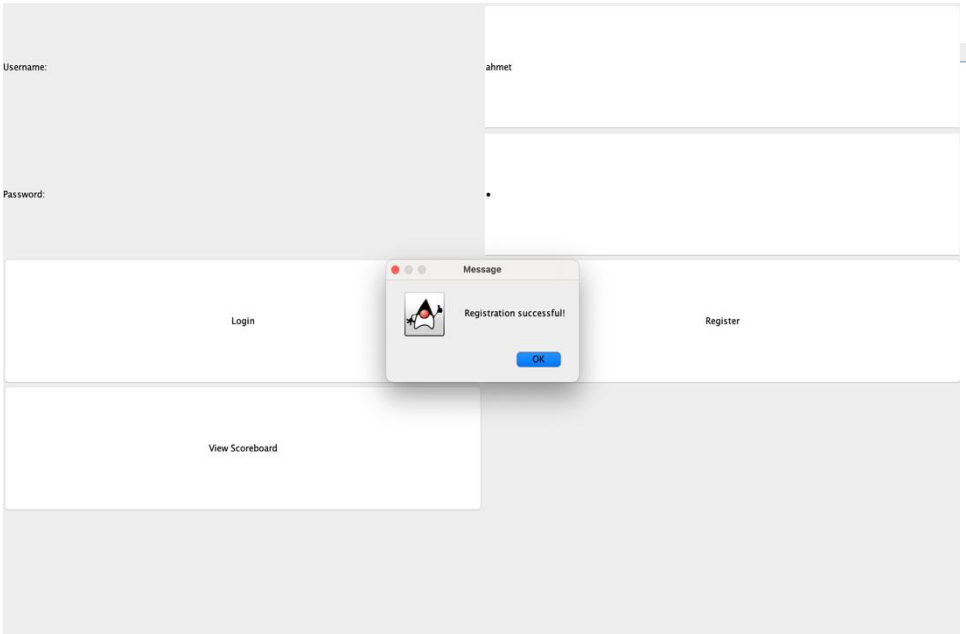
1- If a user tries to log in without being registered, the message **"Invalid username or password!"** is shown..

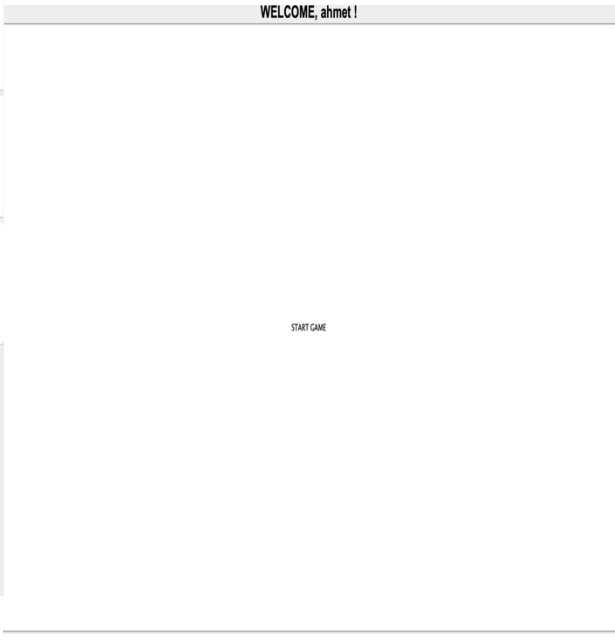


2-If a user tries to log in with a correct username but a wrong password, the system will show: **"Invalid username or password!"**



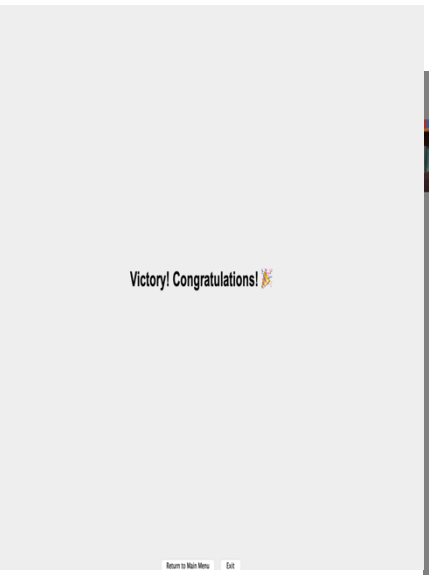
3-If a user registers with a new (previously unused) username and password, the system displays the message:
"registration succeesfull!"
After that, the user is redirected to the main menu and sees the **"Game Start"** menu to begin playing.

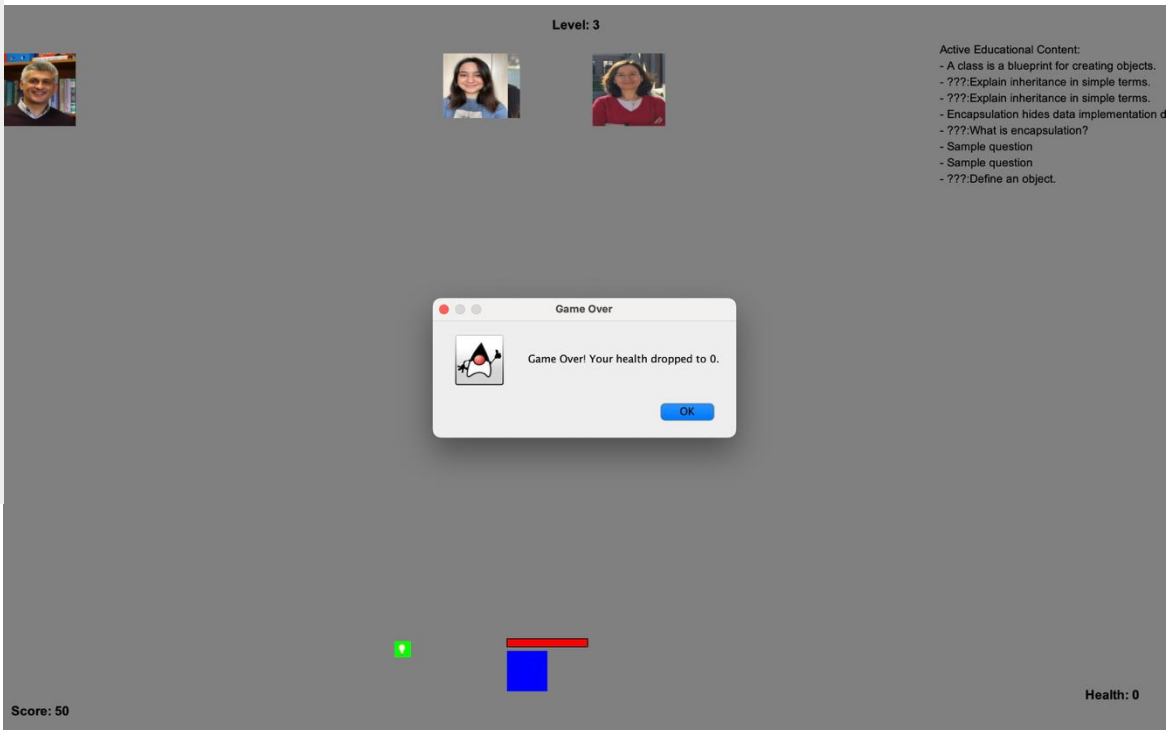


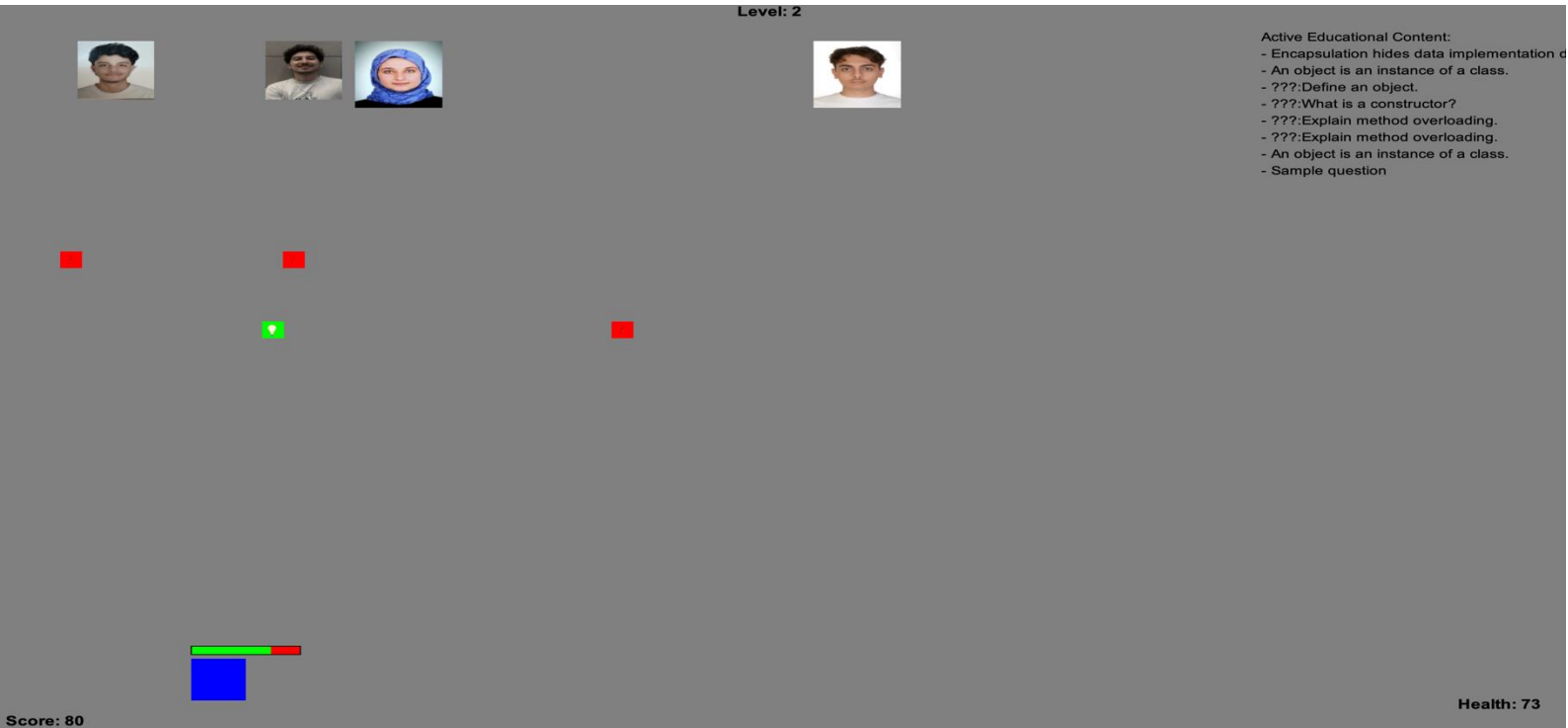


GAMEPANEL SCREEN , WIN OR LOSE

The player controls using the **left** and **right arrow keys** to move across the bottom of the screen. The goal is to **avoid falling question boxes** (and **collect information boxes** .As the score increases, the player advances through levels. Each level introduces new types of **Knowledge Keepers** (SL, TA, Professor), who shoot faster and more complex boxes. Educational content appears on the side .If the player loses all health, the game ends. If the player reaches the required score at Level 3, they win.







Project Design Information

2- Class relations

Core Classes (in core package)

- **GameEngine**
  This is the main class that controls the game logic. It checks things like player health, transitions between levels …
- **Player**
  Represents the player character. Stores health, score, and position. Can move left and right with arrow keys.
- **KnowledgeKeeper (abstract)**
  A base class for all enemies: SL, TA, and Professor. Each one shoots differently depending on its type.
- **SL, TA, Professor**
  These classes extend KnowledgeKeeper. Each behaves differently based on the level. For example, Professors are harder and give more difficult questions.
- **ShotBox**
  Hitting a question reduces health, while information increases score. Their content also shows on the right side of the screen when touched.
- **ShotBoxManager**
  Reuses ShotBoxes instead of constantly creating new ones. This improves performance during the game.
- **CooldownTimer**
  Prevents enemies from shooting too often. Enemies have to wait a short time before shooting again.
- **LevelManager**
  Keeps track of the current level. When the score is high enough, it switches to the next level.
- **LevelGUI**
  A visual label that shows the current level on the screen.
- **ScoreSaverLoader**
  Handles saving and loading scores from the scoreb.txt file. Sorts them by highest score.
- **ScoreEntry**
  A simple class that stores username and score.
- **FileLoader**
  Reads and converts questions and information from files into ShotBoxes. Also loads user data and images.

GUI Classes (in gui package)

- **GamePanel**
  The main game screen. Displays the player, enemies, and ShotBoxes. Handles user input and updates the game.
- **MainMenuPanel**
  Appears before or after the game. Lets the user start a new game or exit.
- **VictoryPanel**
  Appears when the player wins. Shows a victory message and options to return or quit.

- **ScoreboardPanel**
  Displays a list of player scores. Reads data from the score file.
- **LoginRegisterPanel**
  Lets the player log in or register. Shows an error if the username or password is incorrect.

  User System (in user package)

- **User**
  Holds information about a user like username, password, and profile image.
- **UserManager**
  Manages login and registration. Prevents duplicate usernames and checks credentials.

3-Type Hierarchy ,Inheritance,Interfaces and Abstract Classes

This project has a superclass called KnowledgeKeeper. This class contains enemy characters like SL, TA, and Professor. These characters behave similarly in some situation, indeed, have their own unique behaviors. That's why they are all subclasses of KnowledgeKeeper.

Example structure:

- KnowledgeKeeper (abstract class)
  - SL
  - TA
  - Professor

They all have common methods such as move(), shoot(), and getShotSpeed(), but each class can use these methods differently. This keeps the game code organized..

Usage example:

```
for (KnowledgeKeeper k : keepers) {
   k.moveSmoothly();
   int speed = k.getShotSpeed();
```

Thanks to this type hierarchy and **polymorphism**, the program doesn't need to know the exact type of every object—it automatically calls the true version of the method. This makes the code easier to augment with new characters

KnowledgeKeeper defines an **abstract method** called getShotSpeed(), which is used to determine how fast a ShotBox falls depending on the enemy type.

```
public abstract class KnowledgeKeeper {
   public abstract int getShotSpeed();
}
```

Implementation in Professor Class:

```
@Override
public int getShotSpeed() {
   return 6;
}
```

I**mplementation in TA Class:** @Override
public int getShotSpeed() {
    return 4;
}

different knowledge keepers can behave freely without changing the overall game model.This is a clean example of **polymorphism through abstract methods**.

I did not use any interfaces, but in general, interfaces are useful for defining a set of methods that different classes can implement. For example, we could create a Movableobject interface with a move() method, and then have both Player and KnowledgeKeeper implement it and they can use in their own way.

GUI Components and Design

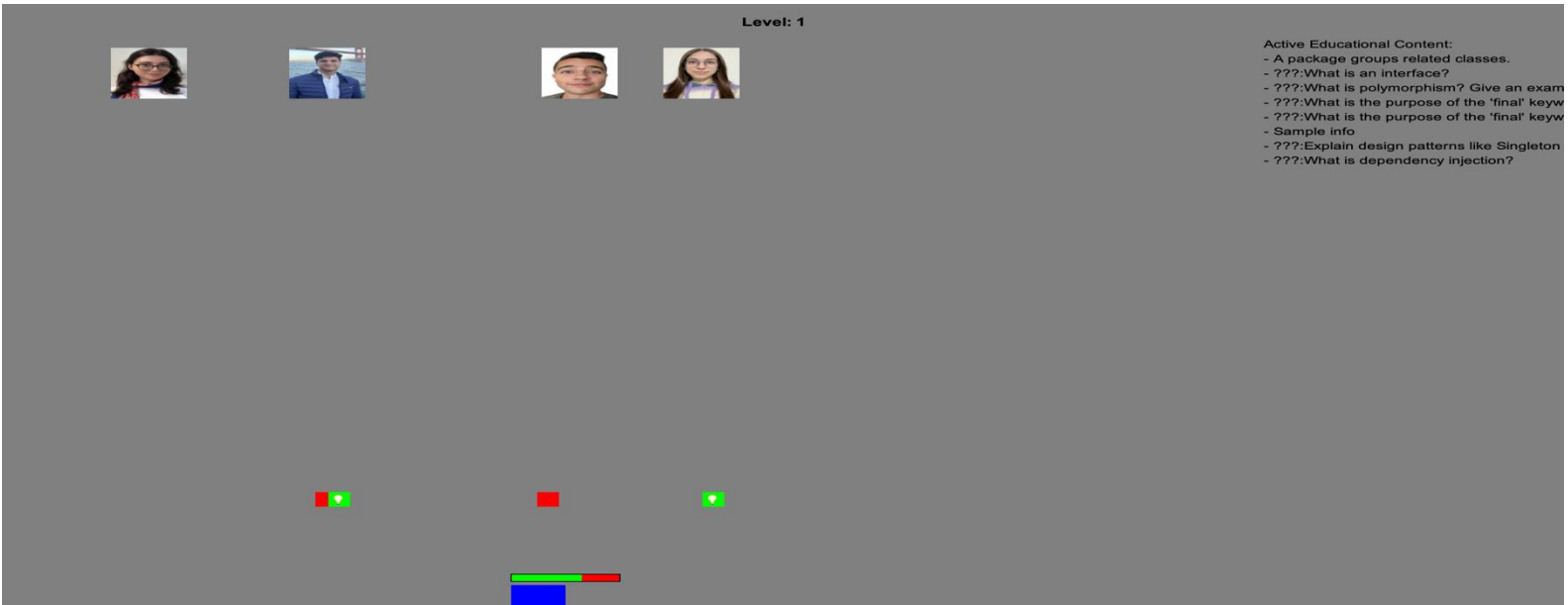
The GUI of the application is built using Java Swing. Each screen, such as the LoginRegisterPanel, MainMenuPanel, GamePanel, ScoreboardPanel, and VictoryPanel, is implemented as a subclass of JPanel. The main frame (JFrame) hosts these panels and dynamically switches between them using setContentPane().

In GamePanel, graphic components are draw by using paintComponent(Graphics g). we call custom draw() methods for game elements the player, Knowledge Keepers, and ShotBox.. Besides this, GUI elements like JLabel, JButton, and Timer are used for user interaction.. Panels are changing between each other. For example, after login, the LoginRegisterPanel passes control to MainMenuPanel, which then launches GamePanel. Similarly, when player wins , it transitions to VictoryPanel.

In the **GamePanel**, we display key game data such as health, score, level, and recent question/info texts using custom drawing inside paintComponent(). These elements help players track their progress and status during gameplay. Also page 3 have some other level and sitation about game .

txt Files Processing Details

In this project, several .txt files are used to manage game data effectively:

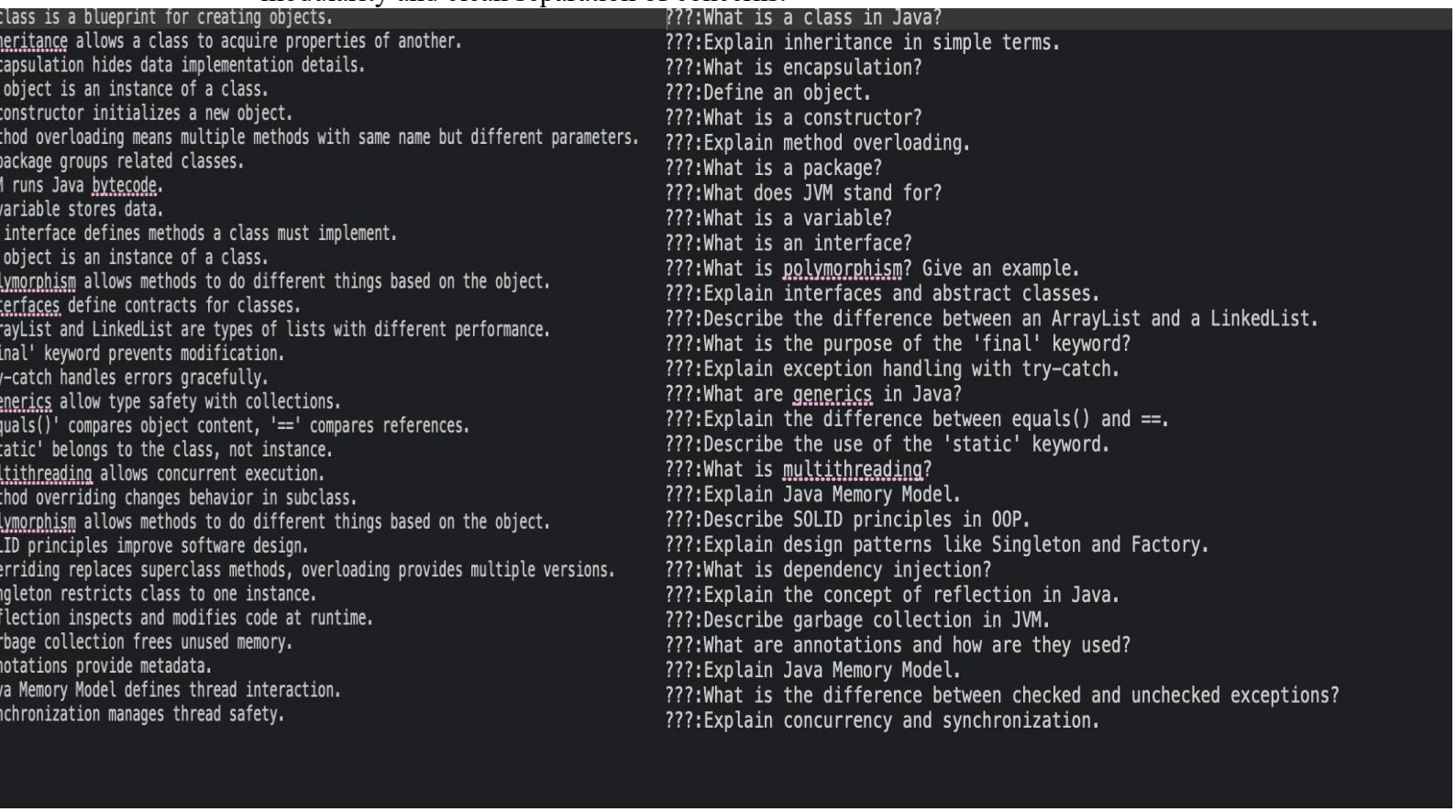


- **users.txt**: Stores user information such as username, password, and profile image path. It is accessed during login and register process.
- **questions.txt** & **info.txt**: Provide the educational questions or infos for the ShotBoxes in the game. Each line in these files is converted into a ShotBox using the FileLoader class.
- **scoreb.txt**: Keeps track of players' final scores after completing a game session. It is read by the ScoreSaverLOAder class to display the scoreboard.
- **log.txt**: Records in-game events for debugging and review purposes.

  THESE ARE INFO.TXT AND QUESTION.TXT

Additional Design Details

- The project is structured into four packages (core, gui, user, and main) to ensure modularity and clean separation of concerns.



- Timers (are used for smooth animations, enemy shooting intervals, and game loop updates.
- Custom GUI components render player health, score, level, and active educational informations during gameplay.

- Due to a technical issue, I was unable to complete Lab 7 on my own computer. As a result, I had to develop the project using a different system. I sincerely apologize in advance for any potential shortcomings or incomplete parts caused by this situation.

References

- Oracle Java Documentation: https://docs.oracle.com/javase/8/docs/
- Lecture materials and slides
- Java Object-Oriented Programming Concepts: https://docs.oracle.com/javase/tutorial/java/concepts/