KOÇ ÜNİVERSİTESİ
COMPUTER ENGINEERING

COMP132: Advanced Programming

Programming Project Report

Knowledge Siege: Simulation Design and Development

<Mert Bayram Köksal>

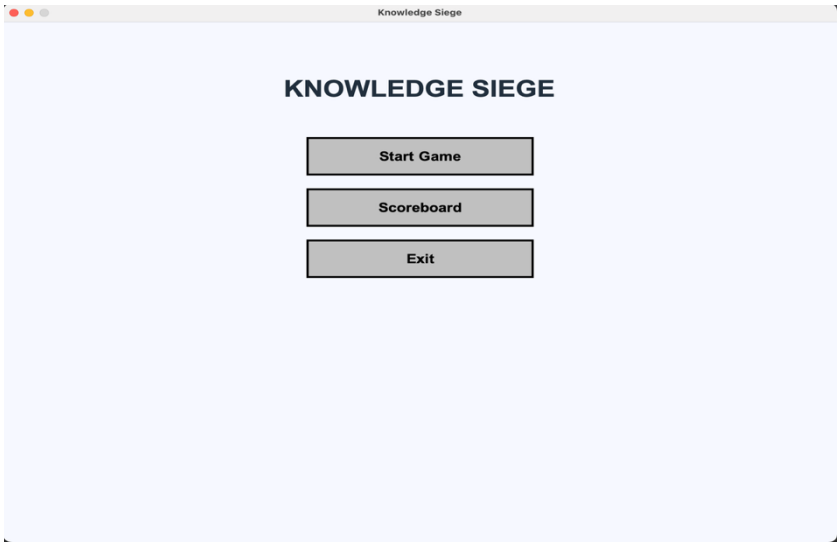
<Spring 2025>

Part 1 – General Demo Information

My game contains the following panels:

- Login / Register Panels
- Menu Panel
- Game Panel
- Scoreboard Panel

When the game is opened, the first login panel appears on the screen. Then you log in, or register if you don't have an account. After logging in, the menu panel appears. From this menu panel, you can start the game, go to the scoreboard panel, and exit the game. Menu Panel:



When you start the game, the game is played. The level is passed by collecting the necessary scores. It consists of 3 levels. If all 3 levels are completed successfully, the game is won, otherwise the game is lost, and the necessary message appears on the screen. Then, the scoreboard panel is accessed. You can also access the menu from the scoreboard panel.

1.1 Pre-Registered Player:

- Username: Mert
- Password: 123

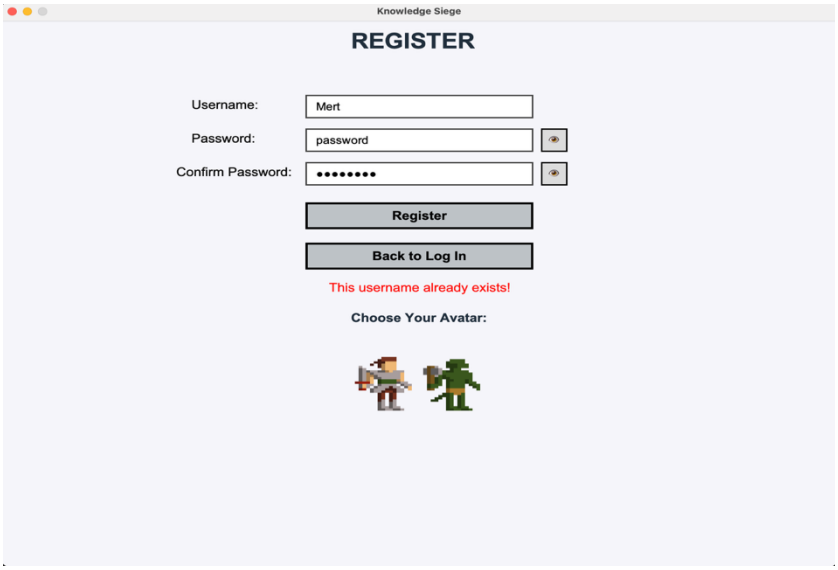
The user enters the username and password in the required fields and then clicks the log in button to access the game. If desired, user can see his/her password by pressing the button with the eye icon.

The logs for pre-registered player:

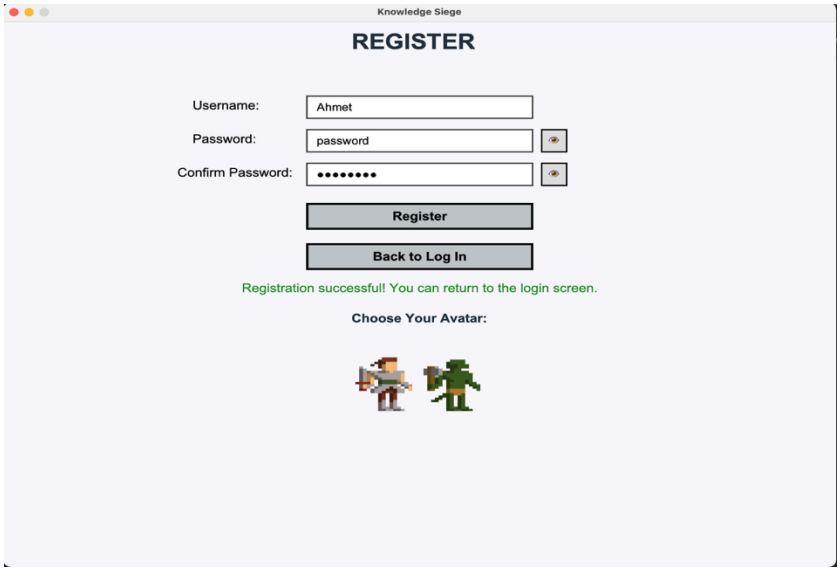
```
[2025-05-27 14:13:24] --------- Game started. ---------
[2025-05-27 14:13:24] Player: Mert has entered.
[2025-05-27 14:13:24] Knowledge Keepers has entered.
[2025-05-27 14:13:31] Mert collected information. +10 points.
[2025-05-27 14:13:31] Score: 10
[2025-05-27 14:13:34] Mert collected information. +10 points.
[2025-05-27 14:13:34] Score: 20
[2025-05-27 14:13:36] Mert collected information. +10 points.
[2025-05-27 14:13:36] Score: 30
[2025-05-27 14:13:36] Mert collected information. +10 points.
[2025-05-27 14:13:36] Score: 40
[2025-05-27 14:13:39] Mert collected information. +10 points.
[2025-05-27 14:13:39] Score: 50
[2025-05-27 14:13:39] ----- Level Transition -----
[2025-05-27 14:13:39] Level 2
[2025-05-27 14:13:39] Knowledge Keepers has entered.
[2025-05-27 14:13:48] Mert collected information. +20 points.
[2025-05-27 14:13:48] Score: 70
[2025-05-27 14:13:49] Mert was hit by a question. -10 HP.
[2025-05-27 14:13:49] Health: 90
[2025-05-27 14:13:50] Mert collected information. +20 points.
[2025-05-27 14:13:50] Score: 90
[2025-05-27 14:13:51] Mert collected information. +10 points.
[2025-05-27 14:13:51] Score: 100
[2025-05-27 14:13:53] Mert collected information. +20 points.
[2025-05-27 14:13:53] Score: 120
[2025-05-27 14:13:54] Mert collected information. +10 points.
[2025-05-27 14:13:54] Score: 130
[2025-05-27 14:13:54] Mert collected information. +10 points.
[2025-05-27 14:13:54] Score: 140
[2025-05-27 14:13:56] Mert collected information. +20 points.
[2025-05-27 14:13:56] Score: 160
[2025-05-27 14:13:56] ----- Level Transition -----
[2025-05-27 14:13:56] Level 3
[2025-05-27 14:13:56] Knowledge Keepers has entered.
[2025-05-27 14:14:04] Mert collected information. +20 points.
[2025-05-27 14:14:04] Score: 180
[2025-05-27 14:14:08] Mert collected information. +20 points.
[2025-05-27 14:14:08] Score: 200
[2025-05-27 14:14:11] Mert collected information. +30 points.
[2025-05-27 14:14:11] Score: 230
[2025-05-27 14:14:14] Mert was hit by a question. -20 HP.
[2025-05-27 14:14:14] Health: 70
[2025-05-27 14:14:16] Mert collected information. +20 points.
[2025-05-27 14:14:16] Score: 250
[2025-05-27 14:14:19] Mert collected information. +20 points.
[2025-05-27 14:14:19] Score: 270
[2025-05-27 14:14:19] Mert collected information. +20 points.
[2025-05-27 14:14:19] Score: 290
[2025-05-27 14:14:34] Mert collected information. +30 points.
[2025-05-27 14:14:34] Score: 320
[2025-05-27 14:14:34] Mert won the game!
[2025-05-27 14:14:34] ------------------------------------------------------
```

1.2 New Player:

To register, click on the register button in the login panel to go to the register panel. Then, enter the username, password and password again, select the desired avatar and click the register button to register for the game. If you try to register with a username that has been used before, an error message will be printed on the screen:



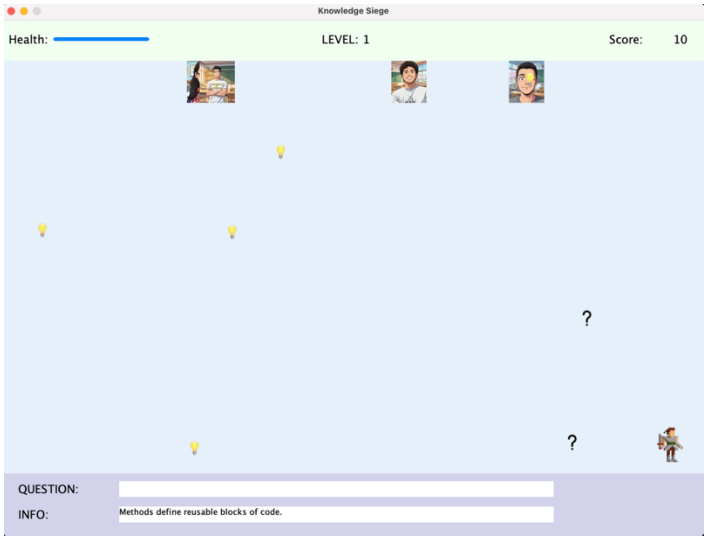
In order to register successfully, all required fields must be filled in and an avatar must be selected. A successful registration can be seen below:

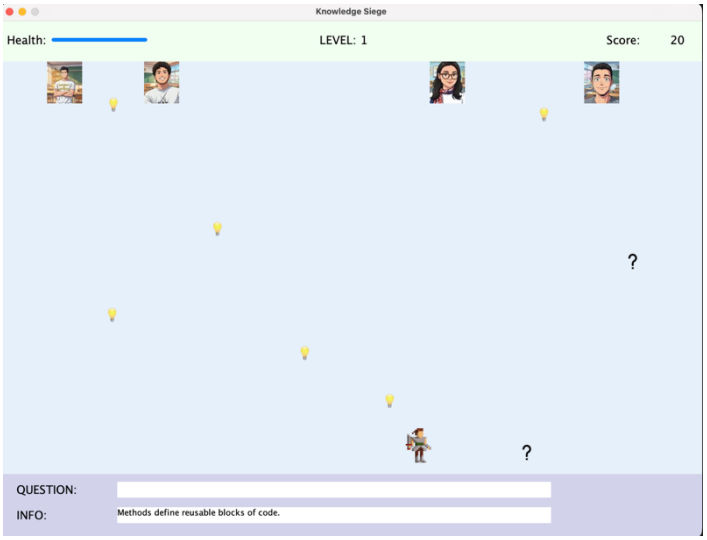


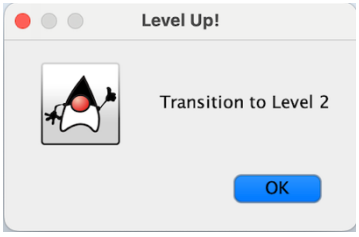
Gameplay:

- The user can move the character with the right and left arrow keys.
- Shot boxes fall down the screen and if they hit a character, the text they contain appears at the bottom of the screen.
- If it hits the question marks, the life decreases, if it hits the light bulb, the score increases.
- Level transitions occur when the required score is reached.

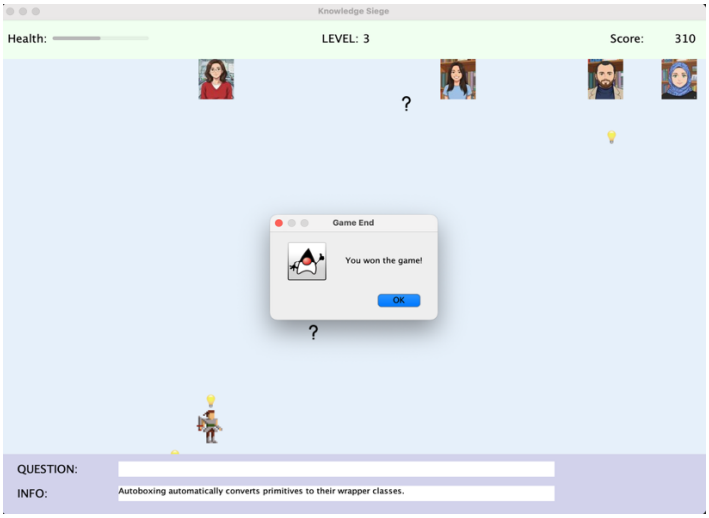
In the images below, the character's movement, the random movements of the knowledge keepers, the shot boxes, the messages displayed on the screen when shot boxes collide, and the level transition message can be seen.

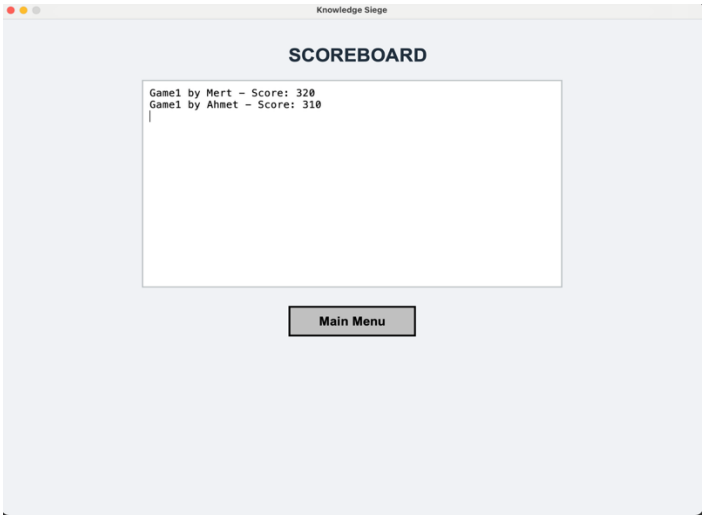






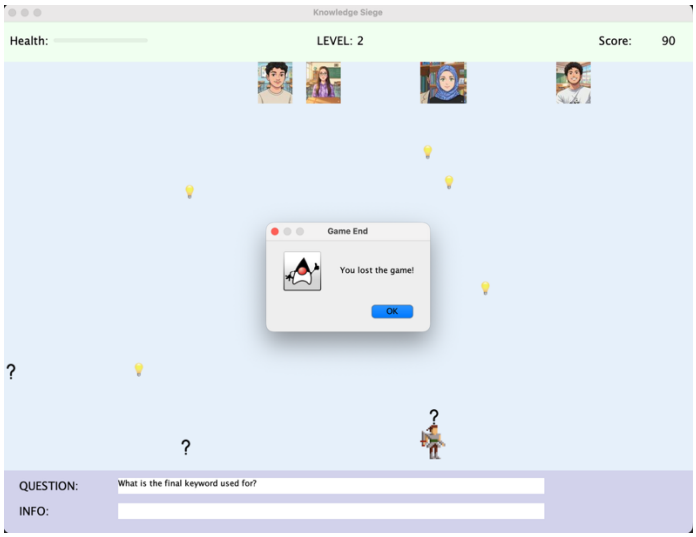
In the images below, the victory screen and the scoreboard that appear when the user is won can be seen.

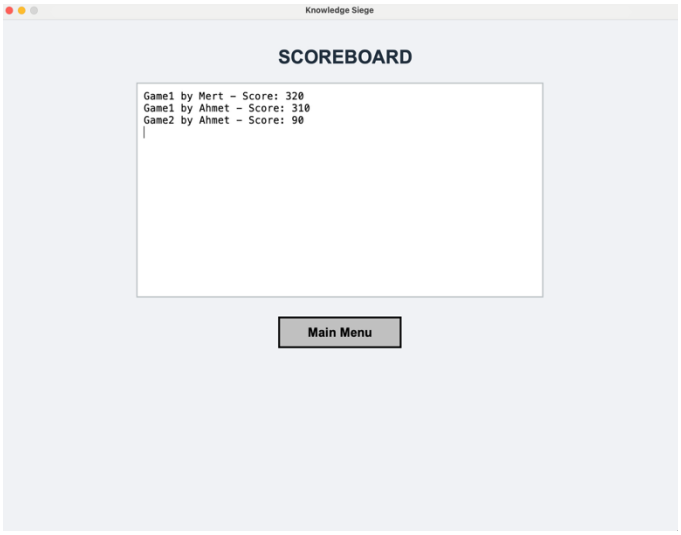




When user successfully complete the game, the necessary message appears on the screen and then the scoreboard opens. The scoreboard is also listed in descending order. If there are two users with the same score, they are listed alphabetically.

When the user runs out of lives, the necessary message appears on the screen and then the scoreboard is displayed. An example of a game that lost at level 2 can be seen in the images below.





The logs for these two games can be seen below. The won game session on the left, the lost game session on the right.

"WON"

```
[2025-05-27 14:14:34] ------------------------------------------------------
[2025-05-27 14:19:07] --------- Game started. ---------
[2025-05-27 14:19:07] Player: Ahmet has entered.
[2025-05-27 14:19:07] Knowledge Keepers has entered.
[2025-05-27 14:19:13] Ahmet collected information. +10 points.
[2025-05-27 14:19:13] Score: 10
[2025-05-27 14:19:15] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:19:15] Health: 95
[2025-05-27 14:19:17] Ahmet collected information. +10 points.
[2025-05-27 14:19:17] Score: 20
[2025-05-27 14:19:18] Ahmet collected information. +10 points.
[2025-05-27 14:19:18] Score: 30
[2025-05-27 14:19:20] Ahmet collected information. +10 points.
[2025-05-27 14:19:20] Score: 40
[2025-05-27 14:19:22] Ahmet collected information. +10 points.
[2025-05-27 14:19:22] Score: 50
[2025-05-27 14:19:22] ----- Level Transition -----
[2025-05-27 14:19:22] Level 2
[2025-05-27 14:19:22] Knowledge Keepers has entered.
[2025-05-27 14:19:28] Ahmet collected information. +20 points.
[2025-05-27 14:19:28] Score: 70
[2025-05-27 14:19:30] Ahmet collected information. +20 points.
[2025-05-27 14:19:30] Score: 90
[2025-05-27 14:19:32] Ahmet collected information. +10 points.
[2025-05-27 14:19:32] Score: 100
[2025-05-27 14:19:34] Ahmet collected information. +20 points.
[2025-05-27 14:19:34] Score: 120
[2025-05-27 14:19:35] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:19:35] Health: 90
[2025-05-27 14:19:37] Ahmet collected information. +10 points.
[2025-05-27 14:19:37] Score: 130
[2025-05-27 14:19:41] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:19:41] Health: 80
[2025-05-27 14:19:41] Ahmet collected information. +10 points.
[2025-05-27 14:19:41] Score: 140
[2025-05-27 14:19:46] Ahmet collected information. +20 points.
[2025-05-27 14:19:46] Score: 160
[2025-05-27 14:19:46] ----- Level Transition -----
[2025-05-27 14:19:46] Level 3
[2025-05-27 14:19:46] Knowledge Keepers has entered.
[2025-05-27 14:19:51] Ahmet collected information. +30 points.
[2025-05-27 14:19:51] Score: 190
[2025-05-27 14:19:59] Ahmet collected information. +20 points.
[2025-05-27 14:19:59] Score: 210
[2025-05-27 14:20:04] Ahmet collected information. +30 points.
[2025-05-27 14:20:04] Score: 240
[2025-05-27 14:20:09] Ahmet collected information. +20 points.
[2025-05-27 14:20:09] Score: 260
[2025-05-27 14:20:23] Ahmet collected information. +30 points.
[2025-05-27 14:20:23] Score: 290
[2025-05-27 14:20:30] Ahmet collected information. +20 points.
[2025-05-27 14:20:30] Score: 310
[2025-05-27 14:20:30] Ahmet won the game!
[2025-05-27 14:20:30] ------------------------------------------------------
```

"LOSE"

```
[2025-05-27 14:20:30] ------------------------------------------------------
[2025-05-27 14:20:39] --------- Game started. ---------
[2025-05-27 14:20:39] Player: Ahmet has entered.
[2025-05-27 14:20:39] Knowledge Keepers has entered.
[2025-05-27 14:20:46] Ahmet collected information. +10 points.
[2025-05-27 14:20:46] Score: 10
[2025-05-27 14:20:47] Ahmet collected information. +10 points.
[2025-05-27 14:20:47] Score: 20
[2025-05-27 14:20:47] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:20:47] Health: 95
[2025-05-27 14:20:50] Ahmet collected information. +10 points.
[2025-05-27 14:20:50] Score: 30
[2025-05-27 14:20:53] Ahmet collected information. +10 points.
[2025-05-27 14:20:53] Score: 40
[2025-05-27 14:20:55] Ahmet collected information. +10 points.
[2025-05-27 14:20:55] Score: 50
[2025-05-27 14:20:55] ----- Level Transition -----
[2025-05-27 14:20:55] Level 2
[2025-05-27 14:20:55] Knowledge Keepers has entered.
[2025-05-27 14:21:03] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:03] Health: 85
[2025-05-27 14:21:04] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:04] Health: 80
[2025-05-27 14:21:06] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:06] Health: 70
[2025-05-27 14:21:07] Ahmet collected information. +10 points.
[2025-05-27 14:21:07] Score: 60
[2025-05-27 14:21:10] Ahmet collected information. +10 points.
[2025-05-27 14:21:10] Score: 70
[2025-05-27 14:21:12] Ahmet collected information. +20 points.
[2025-05-27 14:21:12] Score: 90
[2025-05-27 14:21:24] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:24] Health: 65
[2025-05-27 14:21:26] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:26] Health: 60
[2025-05-27 14:21:29] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:29] Health: 50
[2025-05-27 14:21:31] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:31] Health: 45
[2025-05-27 14:21:35] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:35] Health: 40
[2025-05-27 14:21:37] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:37] Health: 35
[2025-05-27 14:21:45] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:45] Health: 25
[2025-05-27 14:21:47] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:47] Health: 15
[2025-05-27 14:21:50] Ahmet was hit by a question. -5 HP.
[2025-05-27 14:21:50] Health: 10
[2025-05-27 14:21:51] Ahmet was hit by a question. -10 HP.
[2025-05-27 14:21:51] Health: 0
[2025-05-27 14:21:51] Game Over for Ahmet.
[2025-05-27 14:21:51] ------------------------------------------------------
```

Part 2 - Project Design Description

Structure of my project:

- characters (package)
    - KnowledgeKeeper (abstract class)
    - SectionLeader (subclass of KnowledgeKeeper)
    - TeachingAssistants (subclass of KnowledgeKeeper)
    - Professors (subclass of KnowledgeKeeper)
    - Player
    - Shootable (interface)
- data (package)
    - EnemyAvatar
    - InfoManager
    - QuestionManager
    - Logger
- user (package)
    - InvalidUserException (Custom exception class)
    - User
    - UserManager
- shotbox (package)
    - ShotBox
- game (package)
    - GameManager
    - GameSession
    - ScoreManager
- screens (package)
    - LoginPanel
    - RegisterPanel
    - MenuPanel
    - GamePanel
    - ScoreboardPanel
    - MainFrame
- main (package)
    - GameApp (main class)

2.1 Class Relations – GUI components:

characters package:

My characters package contains all the character-related classes used in the game, including both the player and enemy characters. This package includes inheritance, abstraction, interface, and polymorphism features.

The Shootable class contains abstract shoot() method. This shoot() method is a method that every enemy type should override because they all have different shooting styles.

The KnowledgeKeeper class is an abstract class that contains the common features of all enemy types, such as location, height and width., speed, avatar. The class contains the common movement method for all enemy types. KnowledgeKeeper class implements the Shootable interface but does not override the shoot() method because the method different for each enemy type.

The SectionLeader class is a subclass of the KnowledgeKeeper class. It uses the move(Player player, int panelWidth) method of the KnowledgeKeeper class and overrides the shoot() method. This shoot() method create and return ShotBox object.

The TeachingAssistans class is a subclass of the KnowledgeKeeper class. It overrides the move(Player player, int panelWidth) method of the KnowledgeKeeper class and adds extra features, and overrides the shoot() method. This shoot() method create and return ShotBox object.

The Professors class is a subclass of the KnowledgeKeeper class. It overrides the move(Player player, int panelWidth) method of the KnowledgeKeeper class and adds extra features, and overrides the shoot() method. This shoot() method create and return ShotBox object.

Teaching Assistants and Professors follow the player based on a certain probability. I make them decide whether to follow or not every 1 second to create a natural look.

The Player class is a class that contains the features of the character in the game such as location, avatar, health, score, height and width.

shotbox package:

ShotBox class are question or information boxes that KnowledgeKeepers throw in the game. This class contains the location, speed, shot box type, text and image properties of the shot boxes.

data package:

The EnemyAvatar class stores the avatar image paths for the KnowledgeKeeper classes in an ArrayList. Each time a KnowledgeKeeper is created, it randomly returns the image path for the appropriate KnowledgeKeeper type.

The InfoManager class reads the information in the info.txt class and stores the information in 3 separate ArrayLists according to the difficulty level. When the shot box is generated, if information is needed, it returns random information according to the appropriate level.

The QuestionManager class reads the questions in question.txt class and stores the questions in 3 separate ArrayLists according to difficulty level. When the shot box is generated, if a question is needed, it returns a random question according to the appropriate level.

The Logger class records events that occur in the game, such as leveling up, shot box collisions, score updates, and health changes, to the logs.txt file.

user package:

The User class contains the username, password, and scores of users who log into the game.

The UserManager class saves the information of its users in the user.txt file. When the game starts, it gets the user information from the file and creates the user objects. It also stores the user objects in a HashMap. It controls the registration and login processes with the following methods:

```
public void registerUser(String username, String password, String avatarPath) throws InvalidUserException {
    if (users.containsKey(username)) {
        throw new InvalidUserException("This username already exists!");
    }

    users.put(username, new User(username, password, avatarPath));
    saveUsers();
}

public User login(String username, String password) throws InvalidUserException {
    User user = users.get(username);

    if (user == null) {
        throw new InvalidUserException("User not found!");
    }

    if (!user.getName().equals(username) || !user.getPassword().equals(password)) {
        throw new InvalidUserException("Username or password is incorrect!");
    }

    return user;
}
```

InvalidUserException class is a custom exception class for errors that may occur during login or entry processes.

game package:

The GameSession class is a class that contains the username, game number and score properties of a single game of users. This class implements the Comparable interface and overrides the compareTo method. It sorts Game Sessions in descending order of scores. If two different users have the same score, it sorts them alphabetically. compareTo method:

```
@Override
public int compareTo(GameSession other) {
    if (this.score != other.score) {
        return Integer.compare(other.score, this.score);
    }
    return this.username.compareToIgnoreCase(other.username);
}
```

The Scoreboard class generates and returns a formatted scoreboard string by aggregating all users' game scores.

The GameManager class manages the general logic of the game. It keeps the required scores for each level. It checks whether the player can level up or not, updates the required scores when the level is reached, and checks whether the game is over or not. And also, it generates enemies for the current level, starts their shooting behaviour, and returns them.

screens package:

LoginPanel class is the class that represents the game's Login page. This class has JLabel, JTextField, JPasswordField, and JButton GUI components. I used JLabel to display titles like “Username:”, “Password” on the screen. I used JTextField for username input and JPasswordField for password input. I used JButton to log in, to go to the registration page if there is no account, and for password visibility. I also used feedbacklabel for feedback message in case of error or positive result. It contains the following method that performs login control when the login button is pressed.

```
private void handleLogin() {
    String username = usernameTxtField.getText();
    String password = new String(passwordTxtField.getPassword());

    if (username.isEmpty() || password.isEmpty()) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText("Please fill in all fields!");
        return;
    }

    try {
        User user = userManager.login(username, password);
        mainFrame.setCurrentUser(user);
        cardLayout.show(parentPanel, "menu");
    }
    catch (InvalidUserException e) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText(e.getMessage());
    }
}
```

The RegisterPanel class represents the game's registration screen. This class contains the JLabel, JTextField, JPasswordField, JButton, and JRadioButton GUI components. I used JLabel to display titles like “Username:”, “Password” on the screen. I used JTextField for username input and JPasswordField for password input. I used JButton to register and return to the login screen after registration. I used JRadioButton for avatar selection. I used feedbacklabel for feedback message in case of error or positive result. It contains the following method that performs register control when the register button is pressed.

```
private void handleRegister() {
    String username = usernameTxtField.getText();
    String password = new String(passwordTxtField.getPassword());
    String confirmPassword = new String(confirmPasswordTxtField.getPassword());

    if (username.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText("Please fill in all fields!");
        return;
    }

    if (!password.equals(confirmPassword)) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText("Passwords do not match!");
        return;
    }

    if (selectedAvatarPath == null) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText("Please choose an avatar!");
        return;
    }

    try {
        userManager.registerUser(username, password, selectedAvatarPath);
        feedbackLabel.setForeground(new Color(35, 140, 35));
        feedbackLabel.setText("Registration successful! You can return to the login screen.");
    }
    catch (InvalidUserException e) {
        feedbackLabel.setForeground(Color.RED);
        feedbackLabel.setText(e.getMessage());
    }
}
```

The Menu Panel class represents the game's menu. This class contains JLabel and JButton GUI components. I used JLabel to print the name of the game on the screen. I used JButton to start the game, to display the scoreboard, and to log out.

The GamePanel class represents the main screen of the game. I added a JPanel to the top of the GamePanel class. I added the JLabels "Health", "Score", "0", "Level: 1/2/3" and the JProgressBar showing the health bar to the top JPanel. I added a JPanel to the bottom of the GamePanel class. I added the "Question:", "Info:" JLabels and 2 JTextFields for printing the questions and information to the bottom JPanel. The GamePanel class draws KnowledgeKeepers, shot boxes, and the player to the screen with the paintComponent(Graphics g) method. checkGameState() checks the current response of the game. If the level is skipped, it prepares a new level. If the game ends, it calls the necessary method. The setupLevel() method sets the next level. The clearPreviousLevel() method clears the old enemies and shot boxes. Then creates the new enemies. The endGame(boolean won) method also records the player's game and displays the necessary message according to the won variable.

Method that checks the state of the game

```
private void checkGameState() {
    if (!gameManager.isGameOver() && gameManager.checkLevelUp(player.getScore())) {
        if (!gameManager.isGameOver()) {
            JOptionPane.showMessageDialog(this, "Transition to Level " +
            gameManager.getCurrentLevel(), "Level Up!", JOptionPane.INFORMATION_MESSAGE);
            setupLevel();
            levelLBL.setText("LEVEL: " + gameManager.getCurrentLevel());
        }
        else {
            endGame(true);
        }
    }

    if (!gameManager.isGameOver() && player.getHealth() <= 0) {
        gameManager.gameEnd(false);
        endGame(false);
    }
}
```

Methods that set the next level

```
private void setupLevel() {
    clearPreviousLevel();
    enemies = gameManager.generateEnemies(qManager, iManager, activeShots);
}

private void clearPreviousLevel() {
    if (enemies != null) {
        for (KnowledgeKeeper enemy : enemies) {
            enemy.stopShooting();
        }
        enemies.clear();
    }
    activeShots.clear();
}
```

Method called when the game ends

```
private void endGame(boolean won) {
    if (timer != null) {
        timer.stop();
    }
    for (KnowledgeKeeper enemy : enemies) {
        enemy.stopShooting();
    }

    gameManager.getCurrentUser().addScore(player.getScore());
    userManager.saveUsers();

    JOptionPane.showMessageDialog(this,
    won ? "You won the game!" : "You lost the game!", "Game End",
    JOptionPane.INFORMATION_MESSAGE);
    mainFrame.showScoreboard();
}
```

I set it up assuming that only one question text or one information text would appear on the screen. The method that checks if the shot box and the player collide can be seen below.

```
private void checkCollisions() {
    Rectangle playerBoundary = player.getBounds();

    for (int i = 0; i < activeShots.size(); i++) {
        ShotBox shotBox = activeShots.get(i);

        if (shotBox.getBounds().intersects(playerBoundary)) {
            String type = shotBox.getType();
            String text = shotBox.getText();

            if (type.equals("info")) {
                int score = shotBox.getEnemy().getInfoScore();

                player.addScore(score);
                scoreValueLBL.setText(String.valueOf(player.getScore()));
                infoTXT.setText(text);
                questionTXT.setText("");

                Logger.logInfoCollected(gameManager.getCurrentUser().getName(), score);
                Logger.log("Score: " + player.getScore());
            }
            else {
                int damage = shotBox.getEnemy().getQuestionDamage();

                player.takeDamage(damage);
                healthBar.setValue(player.getHealth());
                questionTXT.setText(text);
                infoTXT.setText("");

                Logger.logHit(gameManager.getCurrentUser().getName(), damage);
                Logger.log("Health: " + player.getHealth());
            }

            activeShots.remove(i);
            i--;
        }
    }
}
```

I also make these checks at short intervals using Timer. The method I use with Timer can be seen below.

```
private void startGameLoop() {
    timer = new Timer(20, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            for (ShotBox shotBox : activeShots) {
                shotBox.moveDown();
            }
            for (KnowledgeKeeper enemy : enemies) {
                enemy.move(player, getWidth());
            }
            checkCollisions();
            checkGameState();
            repaint();
        }
    });
    timer.start();
}
```

I provide keyboard settings and enable the user to control the player with the method below.

```
private void setupKeyboard() {
    addKeyListener(new KeyAdapter() {
        public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == KeyEvent.VK_LEFT) {
                player.moveLeft();
            } else if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
                player.moveRight(getWidth());
            }
        }
    });
}
```

The reason why the methods are private is that they are all helper methods and are only used within the class. When the GamePanel object is created, the player is created and the game is started by calling the necessary methods, as seen in the image below.

```
public GamePanel(GameManager gameManager, QuestionManager qManager, InfoManager iManager, UserManager userManager, MainFrame mainFrame) {
    this.mainFrame = mainFrame;
    this.gameManager = gameManager;
    this.userManager = userManager;
    this.qManager = qManager;
    this.iManager = iManager;

    setLayout(new BorderLayout());
    setBackground(new Color(230, 240, 250));

    player = new Player(500, 640, gameManager.getCurrentUser().getAvatarPath());

    setupTopPanel();
    setupBottomPanel();
    setupKeyboard();
    setupLevel();
    startGameLoop();

    setFocusable(true);
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            requestFocusInWindow();
        }
    });
}
```

The Scoreboard Panel class represents the scoreboard screen that appears after the game ends. This class contains the JLabel, JScrollPane, JTextArea, and JButton components. I used JLabel for the "Scoreboard" title. I used a JScrollPane for a scrollable text area and put a JTextArea inside it. Users can see their scores for this area too. I also used JButton so that users can return to the menu.

The MainFrame class is the class that starts the game and controls the transition between panels with CardLayout. First, the old user information, questions, and information are loaded using the UserManager, QuestionManager, and InfoManager classes. Then it starts by creating panels using CardLayout and displaying the login panel. Afterwards, methods that provide transition to other panels are called when the button is pressed in the required classes and the game runs. If an error is caught in the constructor or the startGame(User user) method, the necessary message will appear on the screen and the game will not run. MainFrame constructor and methods can be seen below.

Constructor

```
public MainFrame() {
    setTitle("Knowledge Siege");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(1100, 840);
    setResizable(false);
    setLocationRelativeTo(null);


    try {
        userManager = new UserManager();
        userManager.loadUsers();

        questionManager = new QuestionManager();
        questionManager.loadQuestions();

        infoManager = new InfoManager();
        infoManager.loadInfos();
    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(this,
                "An error occurred. The game cannot be started.",
                "Critical Error",
                JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    }


    cardLayout = new CardLayout();
    mainPanel = new JPanel(cardLayout);


    LoginPanel loginPanel = new LoginPanel(mainPanel, cardLayout, userManager, this);
    RegisterPanel registerPanel = new RegisterPanel(mainPanel, cardLayout, userManager);
    MenuPanel menuPanel = new MenuPanel(mainPanel, cardLayout, this);
    ScoreboardPanel scoreboardPanel = new ScoreboardPanel(mainPanel, cardLayout, userManager);


    mainPanel.add(loginPanel, "login");
    mainPanel.add(registerPanel, "register");
    mainPanel.add(menuPanel, "menu");
    mainPanel.add(scoreboardPanel, "scoreboard");


    cardLayout.show(mainPanel, "login");

    setContentPane(mainPanel);
}
```

Methods

```
public void startGame(User user) {
    try {
        GameManager gameManager;
        gameManager = new GameManager(user);

        gameManager.gameStart();

        GamePanel gamePanel = new GamePanel(gameManager,
                questionManager, infoManager, userManager, this);
        mainPanel.add(gamePanel, "game");
        cardLayout.show(mainPanel, "game");
    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(this,
                "An error occurred. The game cannot be started.",
                "Critical Error",
                JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    }
}

public void showScoreboard() {
    for (Component comp : mainPanel.getComponents()) {
        if (comp instanceof ScoreboardPanel) {
            ((ScoreboardPanel) comp).loadScores();
        }
    }
    cardLayout.show(mainPanel, "scoreboard");
}

public void showMenu() {
    cardLayout.show(mainPanel, "menu");
}

public void showLogin() {
    cardLayout.show(mainPanel, "login");
}
```

main package:

The Main class is the entry point of the Knowledge Siege application. It launches the user interface by creating and displaying a MainFrame instance. To ensure thread safety, the GUI initialization is wrapped inside SwingUtilities.invokeLater(...). This method schedules the task to run on the Event Dispatch Thread (EDT), which is responsible for handling all GUI events in Swing. Using invokeLater is the recommended way to start a Swing application, as it prevents potential issues such as unresponsive UI or race conditions by making sure all GUI code runs on the correct thread. The main class can be seen in the picture below.

```java
public class Main {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
            }
        });
    }
}
```

2.2 Implementation of Save/Load - .txt file processing details:

In the QuestionManager and InfoManager classes, when the game starts, the files required for loading the questions and information are read and the questions and information are separated into ArrayLists according to the appropriate levels. When writing the questions and information to the files, I write "[Level 1/2/3]" at the beginning according to their difficulty levels and separate them accordingly. If an error occurs with the file, the methods throw the error. The methods for loading questions and information can be seen in the images below.

Info Loading

```java
public void loadInfos() throws Exception {
    Scanner scanner = new Scanner(new File("data/info.txt"));
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();

        if (line.startsWith("[Level 1]")) {
            level1Infos.add(line.substring(10).trim());
        }
        else if (line.startsWith("[Level 2]")) {
            level2Infos.add(line.substring(10).trim());
        }
        else if (line.startsWith("[Level 3]")) {
            level3Infos.add(line.substring(10).trim());
        }
    }
}
```

Question Loading

```java
public void loadQuestions() throws Exception {
    Scanner scanner = new Scanner(new File("data/questions.txt"));
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();

        if (line.startsWith("[Level 1]")) {
            level1Questions.add(line.substring(10).trim());
        }
        else if (line.startsWith("[Level 2]")) {
            level2Questions.add(line.substring(10).trim());
        }
        else if (line.startsWith("[Level 3]")) {
            level3Questions.add(line.substring(10).trim());
        }
    }
}
```

In the Logger class, I used BufferedWriter to write the game's events to the file. The reason for using true in the new FileWriter("logs.txt", true) section is to be able to overwrite the old entries in the log file. In the image below, you can see the process of writing to the file in the Logger class.

```java
public static void log(String message) {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    String timestamp = "[" + LocalDateTime.now().format(formatter) + "]";

    File logFile = new File("data/logs.txt");

    try {

        if (!logFile.exists()) {
            logFile.createNewFile();
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(logFile, true))) {
            writer.write(timestamp + " " + message);
            writer.newLine();
        }
    }
    catch (Exception e) {
        System.err.println("Critical Error: Could not write to logs.txt: " + e.getMessage());
    }
}
```

The EnemyAvatar class also takes the file paths of the avatar images for Knowledge Keepers and loads an ArrayList. The constructor of the EnemyAvatar class is called by taking the folderPath. This folderPath changes according to the enemy type. If there is an error with the file it throws the error. The method required to get the avatar paths of the avatar images can be seen in the image below.

```
public EnemyAvatar(String folderPath) throws Exception {
    avatarPaths = new ArrayList<>();
    Path path = Paths.get(folderPath);

    if (Files.exists(path) && Files.isDirectory(path)) {
        DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path);
        for (Path file : directoryStream) {

            String fileName = file.getFileName().toString().toLowerCase();
            if (fileName.endsWith(".jpg") || fileName.endsWith(".jpeg")) {
                avatarPaths.add(file.toString());
            }
        }
    }

    Collections.shuffle(avatarPaths);
}
```

User information is kept in the user.txt file in the as username,password,avatarPath,score1,... When the game starts, the file is read, each line is split by commas, and the necessary data is extracted. Then, user objects are created and stored in a Map. If there is an error with the file it throws the error. After the game ends, user information is retrieved and written back to the user.txt file in the required format. This is how the game is saved and loaded. The methods that perform these operations can be seen in the picture below.

The method which saves user

```
public void saveUsers() {
    try (Formatter formatter = new Formatter(new File("data/user.txt"))) {
        for (User user : users.values()) {
            formatter.format("%s,%s,%s,", user.getName(),
                    user.getPassword(), user.getAvatarPath());
            for (int score : user.getScores()) {
                formatter.format("%d,", score);
            }
            formatter.format("%n");
        }
    } catch (FileNotFoundException e) {
        Logger.log("Users cannot save: " + e.getMessage());
        JOptionPane.showMessageDialog(null, "An error occurred", "File Error",
        JOptionPane.ERROR_MESSAGE);
    }
}
```

The method which load users

```
public void loadUsers() throws Exception {
    File file = new File("data/user.txt");

    if (!file.exists()) {
        try {
            boolean created = file.createNewFile();
            if (created) {
                Logger.log("user.txt was not found, a new file was created.");
                return;
            }
        } catch (IOException e) {
            Logger.log("Error creating user.txt: " + e.getMessage());
            throw e;
        }
    }

    Scanner scanner = new Scanner(file);
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        String[] parts = line.split(",");

        String username = parts[0];
        String password = parts[1];
        String avatarPath = parts[2];

        User user = new User(username, password, avatarPath);

        for (int i = 3; i < parts.length; i++) {
            try {
                user.addScore(Integer.parseInt(parts[i]));
            }
            catch (NumberFormatException ignored) {
                Logger.log("Invalid score for user " + username + ": " + parts[i]);
            }
        }

        users.put(username, user);
    }
}
```

References:

- https://www.w3schools.com/java/default.asp
- https://www.geeksforgeeks.org/java/
- https://stackoverflow.com/questions
- https://www.tutorialspoint.com/java/index.htm
- https://youtu.be/Kmgo00avvEw
- https://youtu.be/ScUJx4aWRi0
- https://youtu.be/U28eKSLI7pw
- https://youtu.be/Su4-yQhzbSw
- https://youtube.com/playlist?list=PL3bGLnkkGnuV699lP_f9DvxyK5lMFpq6U&si=8VZixV60y2AA8p0i
- https://youtu.be/ocb3x0TeoUw
- https://youtu.be/1XAfapkBQjk
- https://youtu.be/OIozDnGYqIU
- https://youtu.be/_nmm0nZqIIY
- https://youtube.com/playlist?list=PLX07l0qxoHFI5MQ-ZWlQ496D25LAlnF8R&si=xtdY0wfBf0Thgmf2
- https://youtu.be/Hiv3gwJC5kw
- https://youtube.com/playlist?list=PLyt2v1LVXYS0_-_nE4ZMfJvhjDlx9kRqL&si=Wo-HBVTwLQTpo2hS
- https://youtu.be/QEF62Fm81h4
- https://www.geeksforgeeks.org/java-util-timer-class-java/
- https://www.tutorialspoint.com/java/util/timer_delay.htm
- https://www.tutorialspoint.com/java/lang/system_currenttimemillis.htm
- https://stackoverflow.com/questions/76769595/how-to-lazy-load-images-in-java-swing