CSE 1325: Object-Oriented Programming
University of Texas at Arlington
Summer 2022
Alex Dillhoff

Project Phase III

Description

In the final phase of the project, you are tasked with building a GUI for the game! Since the model portion of the code has largely been completed in phases I and II, this phase will focus on building up the visual elements.

Features

The features and requirements have some room for creativity. The way individual components are laid out in your application are up to you. As long as you implement the required features, you will receive credit for it.

Main Menu

The main screen of the application should show three core areas: a menu bar, a center panel, and a status bar at the bottom.

Menu Bar

The menu bar should be located at the top of the GUI. You only need to implement one menu with the following options:

- Start Game
- Create Character
- Load Character
- Save Character
- Exit

Center Panel

The center panel will include the same options as the menu bar. These should be implemented as buttons. This panel should also display the player party. In this application, **we will limit the party to 4 characters**.

Party View

The party view should have space to show the names and avatars of 4 player characters. Clicking on one of the party slots should present an option to remove that character from the party OR select a different character that is in memory. The characters that are in this view will be the ones used when the game is started.

Status Bar

A small status bar should be displayed at the bottom of the screen. This will be used to provide information to the user about background operations, player status, and whatever else you find appropriate.

Character Creation

The character creation view will present the user with inputs appropriate for creating a new `Player`. This includes the name, stats, weapon, and an avatar. When allocating stats, this view should present a button to randomly roll stats for them. The user should not be allowed to input more stats than they have available. **All characters should have 15 stat points to allocate as they see fit.**

This view will also include an avatar list so that the user can select what the character will look like in the game. A collection of avatar images will be provided for you to use.

A list of weapons should also be displayed so that the user can pick one for the character. You can use the list we used in past phases.

There should be an option to create the character and one to cancel. Pressing cancel should take the user back to the main screen.

Saving Characters

When the user selects the option to save a character, some view should be presented so that they may select which character (in memory) to save. If that character already has a file associated with it, overwrite the file. Otherwise, prompt the user to select a file to save it to using `JFileChooser`. You should limit the file types to CSV.

If no characters are loaded, warn the user that there are no characters to save.

Loading Characters

Present a `JFileChooser` to the user so that they may select a CSV file to load for the character. Attempt to read the character file. If any errors occur, warn the user. If the player is loaded successfully, provide a message to the user. This can be done either with a pop up dialog or as a message on the status bar.

Starting a Game

When the user starts a game, your application should verify that there are players in the party. If there are no players, warn the users to load or create players first. Present the user with an option to select the monsters that their party will face in combat. They

should be able to add multiple of the same type of monster. There is no requirement to limit the number of monsters, but you may do so if you wish. The user should be able to remove monsters from their list before moving on. Once the monster selection is confirmed, switch to the combat view.

Before the game officially starts, perform any required initialization. This includes rolling initiative for all players and monsters. Random locations should be generated for all creatures. **Make sure that they do not overlap with each other.**

Combat View

Present a grid which displays all creatures using their avatar sprites. Single clicking on any avatar should present a dialog view that shows their name and stats. There should also be a space for a button to end their turn (optional: this space can also include attack and disarm buttons). If it is a player's turn, indicate it on the status bar. In this case, only that player's avatar may be moved on the grid.

A creature may only attack or disarm another creature if they are adjacent to the target. You may choose how to implement the controls for attacking and disarming. Some suggestions are to create a panel to the right of the grid with action options. Another option would be to add the buttons to the information dialog when clicking on a target.

Once a creature falls unconscious (0 HP), you should remove their avatar from the board. The game ends if the party succeeds in defeating all monsters or if the monsters defeat the player party.

The rules of the game will follow previous assignments and projects. An additional rules document will be provided as a refresher.

Bonus: Multi-Threading

You can earn up to 20 bonus points for implementing multi-threading. The bonus points will be split up based on the following:

- (10 points) Implement file operations on a separate thread. Notify the controller when the saving/loading is complete.
- (10 points) Perform the game initialization on a separate thread. This includes randomizing the creature locations and rolling their initiative.

Implementation

In general, your visual elements should not directly depend on any model code or data. Any interactions between the data and view components should be facilitated through a controller. At a minimum, your program should include at least one controller which acts as a delegate between the model and view. You may decide that using multiple controllers to abstract individual components is right for your implementation.

UML Diagrams

Provide a UML diagram that shows the relationships between your model, views, and controller(s).

Submitting

Submit all class files, UML diagrams, and other code files on Canvas as a compressed zip file named `LASTNAME_ID_P3.zip`, where `LASTNAME` is your last name and `ID` is your student ID starting with 1xxx.