CSCI 345–Assignment 3 GUI-based Deadwood
Points: 40

Adding a graphical user interface to a well-designed program is relatively easy. In this assignment, you will utilize the idea of *model-view-controller* design pattern. Your user interface will address two separate things: a visual metaphor that describes the model's current state and a set of gestures (e.g., mouse clicks, drag and drops) which send messages to control program behavior.

Outcomes

Upon successful completion of this assignment, you will

- Understand ***Model-View-Controller*** and other relevant design patterns
- Have an implementation of Deadwood with a *g*raphical *u*ser *i*nterface(GUI)
- Have experience extending an existing software with additional capabilities

Problem Statement

Add a graphical user interface (GUI) to your Deadwood implementation. The idea that mouse gestures and keyboard input are tied directly to the screen is only an illusion. The truth is that input and output are separate program tasks. The Model-View-Controller captures this separation in the View (output) and Controller (input). Since your model is already written (your submission for assignment 2), you will focus on the view for assignment 3.

View

The View is responsible for graphically presenting the current state of the model or part of the model. Views are almost always a *composition* of more primitive UI widgets like icons and text boxes. Your Deadwood view must display the key elements of the game. Your users must know their player's current room. One possible implementation is shown in Figure 1.

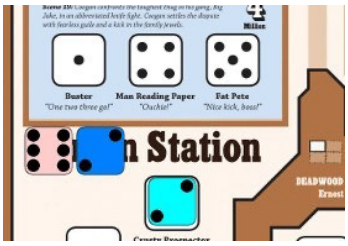

Figure 1. Location of players

Here, we can see that there are three players at the Train Station. We can see that the pale pink and blue players are not working but the cyan player is working. In addition to the player's location, you must also show the player's rank, wealth, credits, and rehearsal points, and you must indicate who the current player is. The rank is already shown in Figure 1. **The amount of money, credits, and rehearsal points must be shown elsewhere.** How you display this information is **up to you**. You could expand the main window enough so that you can include a right-hand or left-hand pane that displays the current user's statistics. This area could also display the current player.

A **shot-counter** is an important detail. Figure 2 shows one way of indicating that a shot was successful and that only two more remains. A day counter, however, is not necessary to display (but you are welcome to display it creatively). You must also show when a **scene has wrapped**. How you show that is also up to you, but it must be obvious.

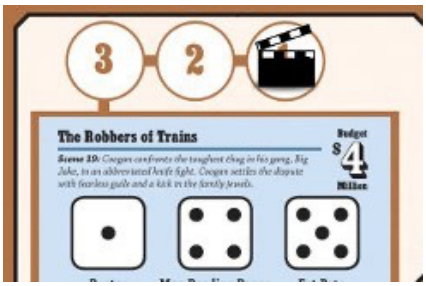

Figure 2: And Action!

Controller

Displaying information graphically is great, but that alone does not create a great program. The user expects to interact with your program through mouse clicks. The question is: how should mouse events be translated into method calls? This is the job of the Controller. The Controller class has UI components, however, the Controller's components are usually invisible. For example, the user indicates a move by clicking on the room's area. For the rooms with a card, this is the card area excluding displayed on-card roles. The Trailer's area is anywhere in the room, and the Casting Office's area is anywhere except for the upgrade costs. Choosing to take a role is simply selecting a role.

Keep in mind that there must be an active card in the room, the role must be available, and the role's level must be no higher than the player's.

While working a role, the user can chose to rehearse by clicking on the card. Alternatively, the user can try to act by clicking on one of the shot counters. If the attempt were successful, it would be nice to block out the selected shot counter, but this is not a requirement. The player upgrades in the Casting Office by selecting the currency for the desired rank. In Figure 3, the user selects 10 Dollars to upgrade to rank 3. The only other action is ending a turn. How a player ends their turn is **up to you**. **Whatever you choose to do, it should be obvious**.

Figure 3: Upgrading the rank

UI Toolkit

You are free to choose any UI toolkit that you like. Use this assignment as an opportunity to improve your GUI design skills or to start learning how to create great GUIs!

Deliverables

You can submit your code via canvas or give us access to your git repository. We will download your implementation from the repository, so please make sure we (TA and myself) have full access to your repository. The file **Deadwood.java** contain should contain the main program.

Presentation

Each student/group will have 5 minutes to present. Students/Groups will show a live demo or a video. During the presentation, students/groups will discuss design and implementation strategy, design and implementation challenges faced and how they

addressed these challenges. For the presentation, points will be awarded based on presentation quality (not implementation).

Grading

20 points Your software correctly plays the game of Deadwood with your GUI-based implementation.

2.5 points Quality of your code (e.g., meaningful comments, well defined methods, descriptive symbols, etc.).

10 points A report specifying which design patterns (or the ideas presented in the design patterns) you used in your implementation and your rationale for using them.

5 points Project Presentation [during class]

2.5 points Contribution Summary (teams only, email individually)

Major Deductions

Case 1: Your program does not compile. If it does not run, we can't give you any point.

Case 2: Your program compiles but break repeatedly. Every time we try to interact with it, it breaks, we won't be able to test the correctness of the program.

Case 3: Your program runs, but does not work correctly. For example, players can move to non-adjacent rooms, players can act in roles above their rank, etc.

Case 4: Your program uses text-based input-output instead of GUI-based interactions.