Assignment 4 Report

Partners: Sahil Chand, Jonathan Peters

Introduction

Jonathan wrote most of the Menu class and the accompanying classes, including the original MenuComponent class and MenuButton class. Our group member Max Bodifee had done some slight refactoring on the MenuComponent and MenuButton class to make the MenuButton extend the MenuComponent class, however further refactoring was required. The Menu class is responsible for loading, drawing, and performing much of the logic of the menu. I noticed that these menu classes were heavily reliant on Enum classes to aid in drawing, loading images and performing the logic, resulting in inconsistencies and reducing the ability to extend the menu class to include more buttons or features.

Code Smells

1. Reliance on Enum Classes

Before refactoring, the Menu class used an Enum Map for the ComponentType Enum class to store its various components, including the Menu's background, its title, and its pay and instructions button. The problem with this was that additional buttons would require additional entries into the ComponentType enums, resulting in inefficient coding. Our solution to this problem would be to create classes to extend a parent MenuComponent class, where each class can store its type, without relying on an enum type to determine what a menu component is.

2. Using Enums to Load Images

The game's ImageLoader class was previously responsible for loading all of the images for the menu's background, banner/title, play button, instructions button, and the corresponding selected/unselected buttons. This was problematic because additional images would have required their own enum value, and the image loader relied on enum-dependent logic to store the images in the menuImages and menuSelectedImages arrays. To get the correct image from these arrays, a developer would need to know the index of the needed array element, which can be frustrating if the menu has many images and many components. Our solution was to create a separate MenuImageLoader class to store the images as variables, instead of in an array. These variables have protected visibility, meaning that the menu can directly reference variables as they are created, aiding the possibility of extension.

3. MenuBG Class

Extending the MenuComponent class, MenuBG is responsible for the menu's background. Its inheritance from the MenuComponent class means that it overrides the draw() and getType() methods.

4. **MenuBanner Class**

Extending the MenuComponent class, MenuBanner creates and draws titles for the menu. This class is distinct because it draws images but not interactable components.

5. **MenuLabel Class**

This class also extends the MenuComponent class, and it was created to eliminate the need to repeat code drawing strings and setting fonts and colors. This class creates a label object that stores a string, font and paint that are used to display the string on the screen.

6. **Long Parameter Lists**

This was a noticeable pattern in this project. It appeared in the Menu Components Classes, and was solved by separating classes into distinct objects that inherit from a parent class. It also appeared in the ImageLoader class, but was slightly reduced with the creation of the MenuImageLoader class. If we made a complex menu system, classes with design templates would be added to create a button object without having to create two or more entirely new images, instead using fonts and text along with a basic image. However, this is beyond the scope of our menu system.

7. **Repeated Code**

The drawSettings method in the Menu class repeated many drawString graphics methods to draw the instructions menu. We chose to use menuLabel objects instead to aid in the creation of label objects while allowing quick changes and modularity. We also replaced the drawString methods when the Game Over screens appear. We concatenate multiple strings to give the player's statistics, and it makes sense to create an object for the entire message.

8. **Reducing Coupling**

Removing the menu images from the image loader class reduces the menu class's reliance on other classes throughout the project. Since the menu runs separately from the gamePanel class, it should not use the same image loader. We also reduced coupling by eliminating the need to call getter methods to extract the needed menu images from an Enum Map.

The menu class now behaves as a cohesive piece of object oriented design.