SWE2410-111 – Lab 2: Checkers Redux – Jawadul Chowdhury – 9/16/2024

Objectives

State your objectives for the lab here: what is the lab supposed to do in your own words. It should be a short paragraph.

This lab is a checkers redux lab. Based on the changes made to the previous lab, the goal of this lab is to reorganize the code so that the strategy pattern is being used within the code. The goal of the lab is to ensure that there is no control coupling and logical cohesion throughout the code. This means changes made in one file does not break other parts of the code.

Requirements

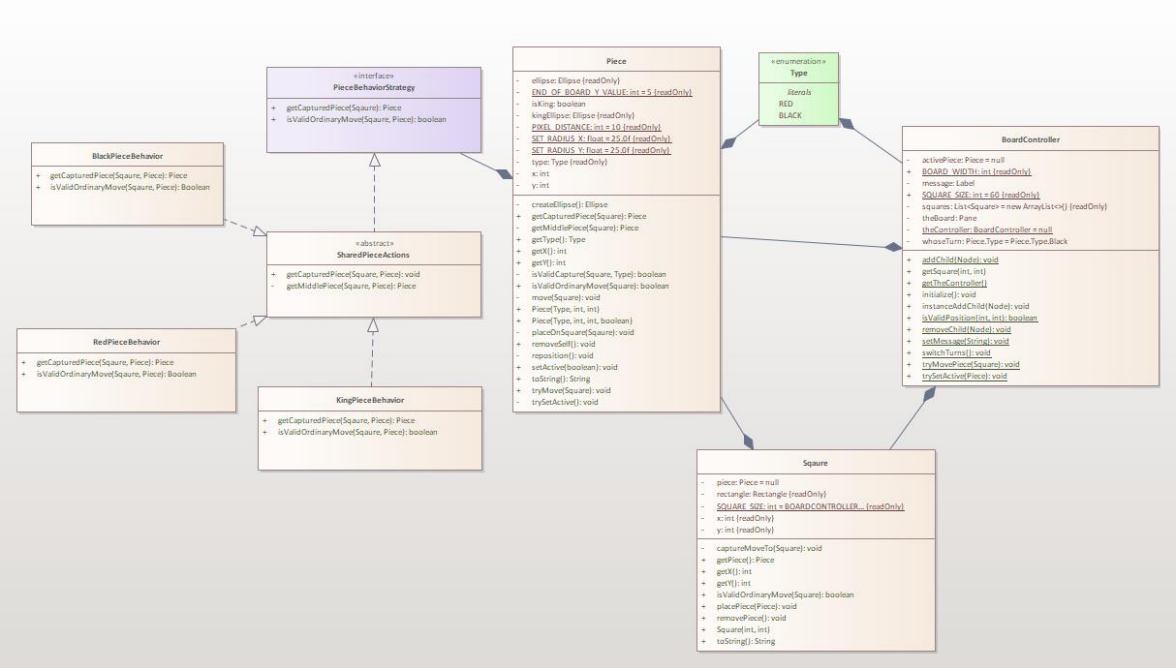
Capture the detailed requirements here. Look at the grading rubrics, the lab assignment instructions to make sure you capture what you must do. Also, put in any requirements that you have added.

The detailed requirement for the lab is listed as follows:

- To create an interface or an abstract class which makes use of methods that would be commonly used by classes of different pieces
- To create classes that for each piece that reflects the movement and behavior of each different piece
- To implement the methods in the classes for each piece, as well as determining how each method will work in each class
- To remove control coupling and logical cohesion from the methods in each class, which means removing if conditions to check for the status of a king, as it is no longer required.

Design

Include appropriate design elements, such as UML diagrams which form the mental model of how it is intended to achieve the objectives. Sequence diagrams are also accepted.



Test Run & Results

What tests did you run to support the requirements of the lab? What was your first pass rate. Demonstrate the outcomes of the tests by providing appropriate screenshots and a short explanation of what the test proves.

With regards to the test runs, all testing was done firstly on the application itself. The tests are conducted to ensure the following:

1. The pieces turn into kings when they reach the end of the either side of the board, and they should have an ellipse stacked on top of it to ensure that it visually represents a king

2. When the king pieces are selected, it should be ensured that both the pieces are highlighted so that the user knows that the king piece is now active and can be moved around

3. The king pieces should be able to move diagonally in any direction, be able to capture other pieces of the opposing color, as well as capture other king pieces of the other color

4. The pieces should have added functionality to ensure that they are able to capture any king pieces in the diagonal direction that they were intended to move, allowing for capturing

Next, we wanted to ensure that our code truly followed the strategy pattern. Therefore, we decided to tamper and change methods within the code to truly ensure that the code followed the strategy pattern:

1. In the BlackPieceBehavior class, the isValidOrdinaryMove() method was tampered with to purposely ensure the code produced a logic error and that the pieces didn't move. The same was done in the RedPieceBehaviorClass and KingPieceBehaviorClass.

2. The abstract class was tampered with, where the getMiddlePiece() method had code changed to produce a logic error. As a result, when the application was run, none of the pieces were able to overtake each other, making it easy to pinpoint where the error was coming from.

Discussion

Conclusions, including what you learned from the tests that you ran.

From running the application, I happened to come across logic errors, where the pieces weren't behaving like I expected them to. Whenever these errors, occurred, they were easy to pinpoint. For example, if a red piece wasn't moving like it was supposed to, then there was a problem with the isValidOrdinary() of the red piece class. The same applies to any logic errors caused by any other pieces.

Any incorrect implementation was easily found through testing on the GUI itself, and once these errors were caught, they were very easy to pinpoint due to how the code was separated into different strategies, instead of one whole class, where other strategies would interfere with each other. Rather, the strategy approach ensured that all strategies are separate from each other.