Jerome Mueller

CPSC 330 – 10/9/13

Master-Mind Project Report

All the members of our group decided we wanted to go above and beyond for this MasterMind project. Specifically, we wished to implement a graphical interface for the game. This decision led to some slight turmoil during the planning stages, but ultimately led to a more reasonable division of work and a better looking final product.

Before our first meeting, we all had reviewed the project guidelines. Russell and Zach had already started toying with ideas by writing segments of code. By our first group meeting, we were surprised to find that between their code snippets, we could have created a rough, but functioning program. Because we still had a large amount of time before the project was due, we decided then to add additional features to our program, namely a graphical representation of the game board. I was to be responsible for creation of the graphics, since I had prior experience with Java graphics. Russell was given the responsibility of writing a Code class which would store codes, validate them, and return the number of correct places and correct colors when comparing codes. Zach was responsible for combining the code logic and the graphical interface into a controlled and working game.

The design we implemented contains five classes. One is the Code class, which is responsible for storing codes. The codes are stored as a string. Upon creation, each code is checked to see that it contains four characters, and that it is contained within a specified alphabet whose size is represented by a static variable. The Code class also converts all codes into upper case. The graphics are split into two classes, one for the frame (MasterGUI) and one for the board (MasterMindBoard). The frame contains an input JTextField and a submit button, as well as the MasterMind board. The board is represented as its own object, and interfaces solely with the MasterGUI object. It contains strings and numbers representing previous guesses as well as their results. These previous guesses are displayed on

the board. It will also display the correct code at the end of the game. A class Game prompts the user for the initial game parameters, then actually plays the game. Finally, there was an ActionList class which serves the sole purpose of registering when the user hit the submit button on the user interface. The Game class uses an object of this class to determine when it should obtain the user's text and compare it to the correct code.

The two modular parts of the code, the GUI and the Code class, worked extremely well on their own. We found using both simultaneously to be somewhat more difficult. We had slight trouble converting the user-entered text into a validated code object. We also had significant trouble determining exactly how pressing the submit button would cause the Game class to act. Hopefully, more experience with event-driven programming and working as a group will alleviate these problems in the future.

We feel that we have built an enjoyable product. The addition of graphics added in a large potential of unforeseen difficulties and complications, but we paced ourselves through the project and completed it.