Software Extension Retrospective

Steven Deutekom
CPSC 3720
University of Lethbridge

December 5, 2019

1 Introduction

Every developer will at some point in their career have to work with other peoples code (probably this is all they will do). So it is important to understand how to read and understand other developers code. It is also important to make one's own code easy read and extensible so that it is easier for future developers to work with. This assignment has given me some insight into the process of working with another developers code. Taking another authors code and adding additional functionality went relatively smooth, but it was not without its challenges. This assignment has been a valuable insight into working on other's code, and how my choices will affect others in the future.

2 The Easy Bits

I found that the code I chose was reasonably well set up for extending. The class design allowed me to reuse every class and just add the functionality that I needed for specific tasks. Virtual functions in the controller classes allowed me to easily override certain functions to handle the cases specific to the game Rummy. The main game loop for the Game class was setup to use these virtual functions in a sensible order. In the end I did not have to modify the code I was given at all. With a little more time I probably could have made some better design choices with my own code, but this was not because of any flaws in the code I used. I was quite pleased that I did not have to re-write anything that was already included, which felt like a victory for me, but was probably more of a victory for the codes original author. There may have been issues in the Old Maid specific code, but the code I worked with for making Rummy made things pretty easy.

3 The Challenging Bits

For me the main challenge was that I wanted to create new sub-classes that needed some functionality not present in the parent classes. The new functionality was not able to be covered by the parents interface. Since my custom features would only be needed in my controller class it did not break the interface for being used in parent class implementation. However, since the parent classes saved pointers to the base class new functionality was not available in my controller.

For example I created a Rummy deck that added a discard pile to the original deck class. The Game class takes in a Deck* and my rummy deck would work properly in Game functions. However, in the Rummy class that inherited from game RummyDeck specific functions needed to be called. I am not entirely sure how people solve this problem under normal circumstances, but I really did not want to cast my deck pointer to a rummy deck pointer anytime I was goint to use the rummy deck. To get over this my RummyDeck constructer takes in a RummyDeck pointer. This seems OK since to play Rummy the deck must be a rummy deck. Then in the Rummy class I saved a pointer to this deck that was a RummyDeck pointer. This means that I am saving two pointers to this deck. However, given that there will only ever be one instance of Rummy when playing the game, and a single pointer does not add much memory. Doing this also allowed me in the Rummy class to always access the deck as a Rummy deck and have access to all it's methods. This solved the problem very nicely and made working with my deck in Rummy very natural. I also used this pattern with my RummyUI as it also needed several distinct methods not in GameUI.

As mentioned above there were not many challenges. the code was well designed and I did not have to work very hard to re-use it. It is always a bit of work to understand and think about how to use another person's code, but this code made it easy.

4 Code Quality

The overall code quality was good. It was well documented and easy to read. The classes and methods were focused and didn't try to do too much. The only real issue I ran into was that the Player class had a score variable that was private, but no getter and setter were provided. However, it was easy to add. All the necessary functionality was tested and worked as expected. This made it easy to rely on the code without fear that it would cause problems. The testing of the Game's start method was especially helpful because I did not have to write a test to play a whole game for me. I was able to just write and test the methods used by the start method.

5 What Would Have Made My Life Easier

As I have said a few times there was not much that made life hard. I think it would have been nice if a few more of the functions in Game were virtual. I understand why they were not, but it could have allowed a little more freedom when sub-classing to be able to add behaviour to these functions. There was the method hasSet in Game that deleted cards when it was done with them. This method was not usable with Rummy for this reason. If instead of deleting the cards it had returned them OldMaid could have deleted those cards, but rummy could put them onto the table. In the end I opted to validate sets and runs players wanted to put down instead of automatically removing them, so it was not a roadblock for me. However, it would have made the code a little more extendable. I think the code did a pretty good job of allowing for additional functionality without trying too do to much.

6 What Would Make Life Easier For Future Devs

It is hard to say at this point, but I think that I did a pretty good job with the original old maid project. I tried to make all of my model classes do what they did very well, but still be easy to add to for future games. I don't think I actually implemented a method to play a game, but I tried to make an interface that would allow the breaking up of turns. I could probably have thought a little more about what kinds of additions that people might need to make when adding more games. But, as I think lot's of people found, no amount of forethought can anticipate the more specific needs of a game.

The extended code's Game's start function had a beforeTurn and afterTurn with some code to get a card from the player. In my game I used these, but the bulk of the players turn for rummy was put into before turn. This works well, but it starts to feel a little strange to call a method before turn, when it is really the whole turn. I also broke mine into a number of rummy specific methods for testing, and they could possibly be extended by other games, but they were very rummy specific. It is quite possible that this algorithm would not be good for as many games as could be desired. However, I think the most important thing is that the model classes are really easy to use and add to.

In the future I would just make sure to spend a good amount of time making sure that my model classes covered all basic functionality that might be needed. That way they could be inherited by or aggregated into model classes that needed a little more functionality.

7 Other Thoughts

I think that I did a good job re-using the code given to me without making any changes to it. I was still able to make a rummy game that worked well enough. Any thing that I would like to have done better with my Rummy implementation could have been done with what I was given. Some of it did take a little though, but it was worth it. In the end I had to implement less than I had to when originally building my own Old Maid. I also had to test less because a lot of the functionality was already tested. I think that in many situations code that is relatively modular can be re-used in some way. It may require some use of design patterns to adapt classes to work together, but it can be worth it if the re-used code is of a high quality. Not only does it save time and money in development costs, but it can eliminate the work of solving difficult problems that are already solved.

8 Conclusions

I appreciated working on this project. I think I learned a little about what it takes to work on another persons code. It may have been more instructive to re-use a project that was not so easy to extend. The biggest questions arise when you can use some parts, but other parts are not quite right. Deciding how you go about that is interesting. Do you just add the parts you want to the classes or functions you have? or do you try to build new abstractions that take advantage of as much as possible? or maybe you just re-use what fits really well and build your own solutions to everything that is not easily extendable? Like all of the lessons we learn in software engineering there are benefits and trade off to all of these approaches. Sometimes no matter how much you want to code has become overly complex and messy from being maintained for a long time. There is a time when new solutions need to be created. There is probably a line where if you have to work too hard to integrate old code into a new project you are not saving yourself much time and money in the future. So being mindful of these things can help a person to think about what can make their code as re-usable as possible. While one can still keep in mind that no matter how well they write a piece of code or design a system it will never work everywhere. One just has to find the best balance of these things in each individual situation.