Stephen Townsend
CS362-400
10/8/17

Refactoring

In order to refactor the 5 cards for Dominion, I started with the first two cards that were required. First, I created a function above the cardEffect function named [cardname]Refactor. I then took the existing code that was in the cases and copied it into the cards function. In the case I placed a call to the function. I now needed to figure out what information the function needed.

I started from the top of the code and worked my way to the bottom. I went through each line. Each word, I checked it to see where it came from and if it was created and initialized in the effect function. If the item was created in effect then I needed to check the purpose. I noticed that if it was an item such as int I, I would be able to create the same item without having to pass it along. The only exception was int z because it was specifically a counter. Since I can make the assumption that this counter will be used by other cases and the total may be important. As a result, I decided to pass the address.

The next step was to look at the what was being passed into the cardEffect function. For example, struct gameState *state was passed in and this needed to be passed to each of the refactored functions. Once I found all of the items, that needed to be identified in the new function, I took that information and used it in the function call. Once I finished a function I would upload the saved code to the server compile it to make sure the program still functioned. If there were any error then I would have to address it.

I followed this process for the next four cards. The cards that I decided to refactor were smithy, adventurer, council_room, feast, and mine. There was no particular reason for the cards I chose besides what was required. I just chose the cards that were at the top of the cardEffect case list.

Bugs

For the bugs, I had a few options. I decided not to have the program crash, but instead switched some information around. For example, with the mineRefactor function call I switched the positions of choice 1 and choice 2. This would make it so that these two items would be swapped when they went into the function.

The next bug that I implemented involved all the function calls besides feast. I noticed that each case is designed to return something when it is complete. Well this is now changed because instead of saying return (functioncall) I left the return off. Each function is called and only returns within the function.

The next bug is in the functions for smithyRefactor and adventurerRefactor. If you look at the functions that come after, each function is of type int and they return an int. I left off the int type for both of the functions and they still have a return at the end of the function.

Here I will talk about a few bugs. In the council_roomRefactor there is a for loop that should count to 4. It now counts to 5. Instead of adding the +4 cards, it will now add +5. The for loop in smithy has changed from 3 to handpost. The z counter in the else statement in adventurerRefactor went from z++ to z = 0.

Stephen Townsend
CS362-400
10/8/17

The card that I chose not to have any intentional bugs is feastRefactor. It was the fourth card I implemented in the refactoring processes.

Summary of bugs:

Smithy – Missing function type, No return = [functioncall], For loop is handpost from 3.

Adventurer – Missing function type, No return = [functioncall], z counter from the else statement went from z++ to z=0.

Council_room – No return = [functioncall], For loop is a 5 from a 4.

Feast – None Implemented.

Mine – Choice 1 and Choice 2 switched in function call, No return = [functioncall].