Brandon Swanson

Week 6 Assignment Report

CS 165 Summer 2014

Understanding

This week we continued on learning about classes and were tasked with making a larger program that is able to wrangle separate encapsulated parts. Namespaces are an interesting method for avoiding naming conflicts especially when working with other developers. When developing my library files I had been putting my last name in front of the class names for similar motivations. But using the namespace seems to give even more protection against this problem, beyond just novel class names. The other benefit for me was it made the function calls in my code that referenced libraries of my own creation glaringly obvious.

I think that at this level the learning objectives of multiple step, separate compilation is purely abstract. I have witnessed others working in a production environment where building (compiling) can take considerable time. But even the largest programs I have written up till now take un-noticable moments to compile. But just because that is the case now doesn't mean It will always be so I tried to pay attention to the mechanics of object file compilation. This process shed some light on how the c standard libraries we use are referenced in our source code and compiled. Via a similar process it is not possible for us to have the original source code for libraries such as <iostream> and <string> but by including the headers we have access to their interfaces.

As requested in the Exercise Components this week. Here is a description of multi-complilation in my environment:
I develop in Eclipse but compile in terminal using g++. There is a build and run option built into Eclipse, but after learning to compile using g++ in terminal I have preferred to continue doing so.

To make a project with separate files and namespaces is simple. To create a namespace for

the functions that I create I must surround a block of code with a namespace definition. ie namespace foo{ }.

To reference these later a using directive for that namespace can be placed in my main .cpp file. Or preferably a using declaration for specific functions I want to use from that namespace. Similar to specifying using std::cout. Thirdly I can use the scope resolution :: for every instance of the function being invoked.

For multi part compilation in g++ I must only include the -c flag to compile separate .o object files. This could help save time when working on very large projects when going through short cycles of changing-compiling-testing. Via this method (especially when combined with the Make utility) I could only re-compile the object file that needs updating. After having these separate object files compiled I can use these as arguments in the g++ call to link these object files together into a compiled executable.

Design

The request this week was to turn our previous game assignments into classes. As I had already done this in previous weeks I wanted to push myself to do something more with classes. So I looked at the common processes that were executed by each of these guessing games and tried to develop and abstract class that each of them could be derived from. It was made into a template class so that the game could be played with a number as a secret and with a string. One of the more difficult decisions was how to handle the input. I decided to have the input returned as a string, and then the abstract function ValidGuess() would be responsible both for converting it to the appropriate type and checking its validity against game rules. This seemed the only way that would work in all casses because a string can always store the input given by the user.

I went back and forth for a little while on the structure of the Phrase Game and Word Game. It was obvious that they were very similar and should be able to share some implementation but not

obvious which should be the parent class. I had a working version where the Word Guess was the parent class, and attempted to keep its methods very general. But then as I made changes and was adjusting the Word Guess methods to an increasing level of generality I realized that the two should both be children of an additional abstract class. Now there is a StringGame abstract class that is a child of the first abstract class and implements some of its methods. WordGame and PhraseGame only need to override two methods (GenerateSecret, and ValidGuess) in oder to function seperatly. Below are diagrams of these two changes along with a diagram of the designed flow of GuessGame::PlayGame()

I was also able to implement some of the designed Menu Item subclass I had envisioned last week. Both the Boolean item and the List item. These made for a menu that could modify different settings without needing a sub menu system to navigate. See the difficulty and computer/human generated for examples of this.

I also chose to keep the difficulty settings contained within my controller class. The assignment requested that our games run self contained. I think this is a great example of the power of classes. By designing the classes with sufficient setters I am able to change the definitions of Easy/Med/Hard from outside the classes.

Testing

Fortunately with these repeat games I was able to use the test cases from previous weeks to establish that the rules of the game and the input handling were functioning as expected. A lot more testing had to go into making sure that classes were being constructed properly and that fields were getting properly initialized.

The GDB debugger was of great help. I had only just learned how to tail a file in Linux and was very excited to use that for logging messages when this week we learned about the GDB. I still want to try to use file tailing with clog << statements but this week as the games were not being developed from scratch I did not end up needing it.

However with the GDB I was able to much more quickly track down the source of nefarious bugs. Using its functions like “where” and “display” I could look into the inner machinations of my program. I had one bug where I had changed the method in which I built the phrase reveal string and was attempting to replace characters in a string that had not been initialized. I was operating with false assumptions but by displaying the value of various fields with the GDB I was able to discover this.

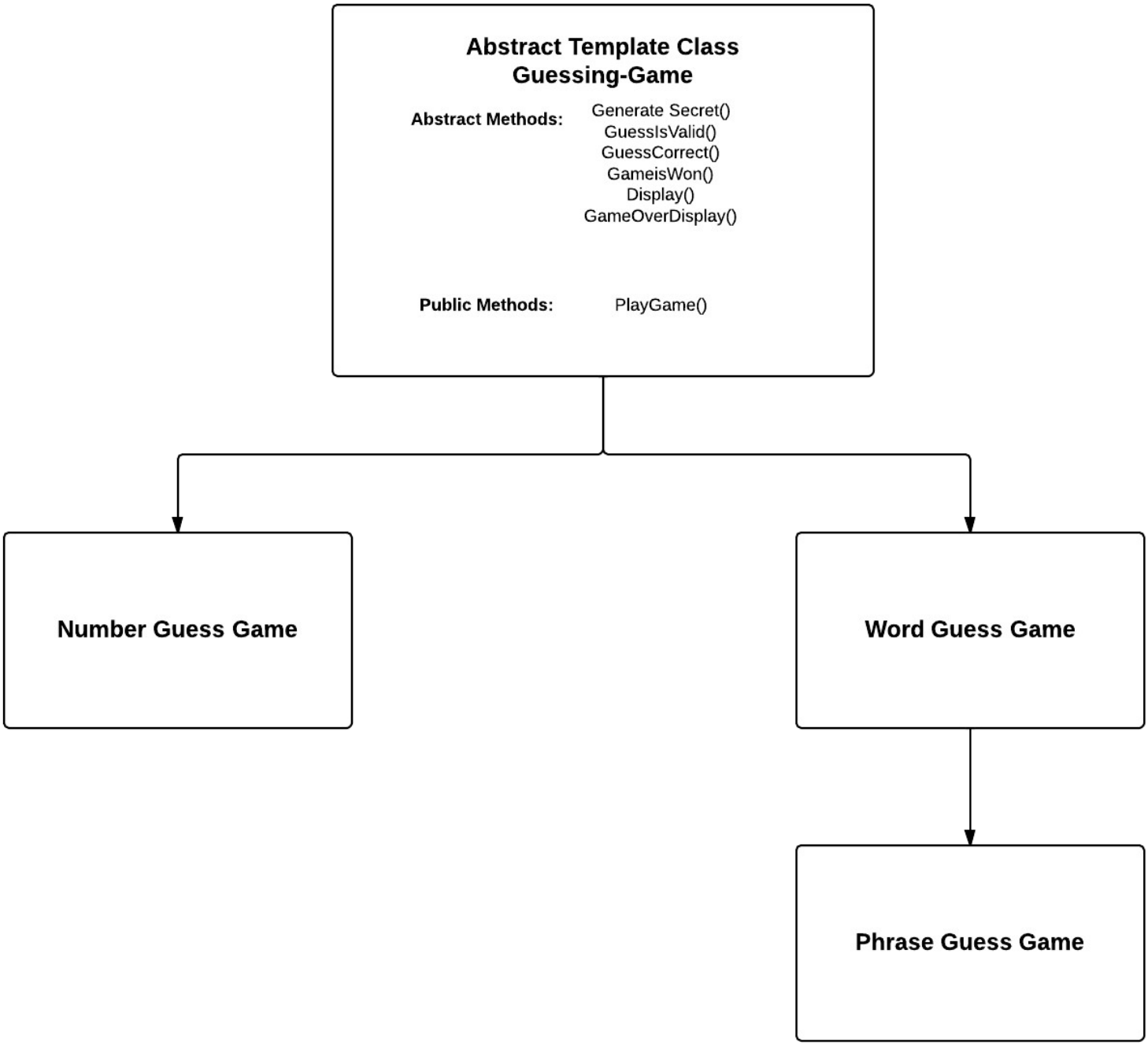
Reflection

Its very easy to see the advantages of developing in an object oriented manner. These games can be easily modified and interacted with. And many of the objects I used in this weeks assignment demonstrate the portability of well designed objects. The Dictionary and Menu class are both 4 week projects now, porting easily from one project to another. Their flexibility is well demonstrated also as I have been able, with great ease, to make modifications or expansions on these objects. Particularly with the Menu class. As each different kind of Menu-Item subclass can continue to add functionality without exposing the core of the object to any revisions, reducing the chance of introducing new bugs.

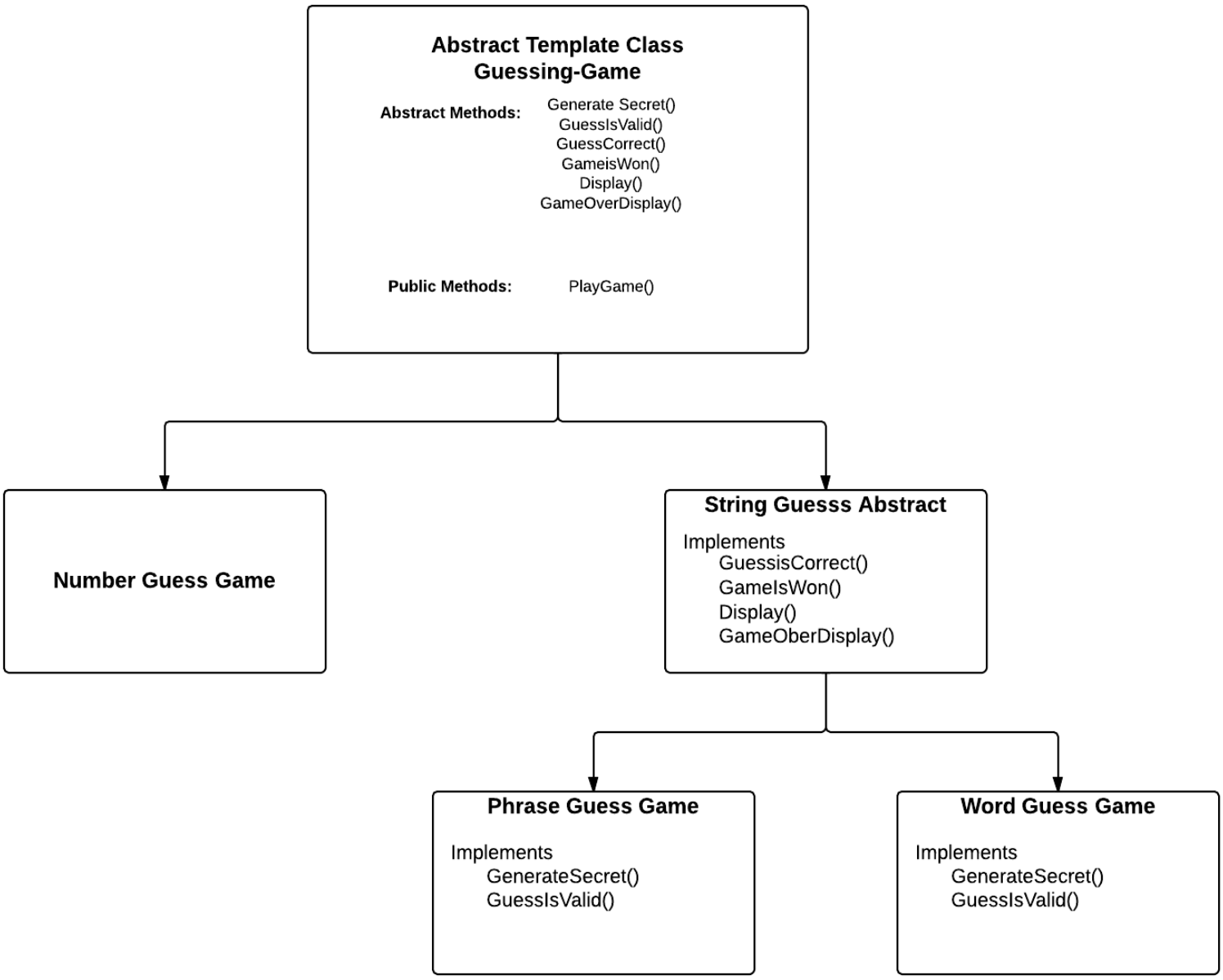
That being said. I am continuously challenged by the current implementation that relies on a call to a function with no parameters and no return. I at first wanted to be able to use the PlayGame() function as the parameter for the menu items, but was unable to do so because even though it matched the signature void() it is a member function and this is different from a normal function pointer. I also attempted to build a specialized menu item just for handling these game objects but because they are implementations of a template class the menu item would also have to be a template class. But the menu object has a list of pointers to MenuItems not template class MenuItems.

As of now this is still the best implementation I have been able to design so I will stick with it despite the limitations, While searching for something more flexible.

Abstract Class First Design



Abstract Class Re-Design



PlayGame() flow design