Brandon Swanson

August 30h 2014

CS 165

Final Project Report

see readme.txt for a list of command line arguments and program controls.

Program Description

I decided to take the classes and interfaces I wrote for the game of life and adapt them to an ncurses application. I wanted to explore using the functionality of the ncurses library, primarily the ability to more easily output characters in a grid system fashion, and to have access to unbuffered keyboard input. More importantly I wanted to put the design plans of my game of life program to the test. It was the most ambitious OOP design I had undertaken this term and the goal was to have the workings and interactions of the systems abstract enough that different output methods could easily be used or the data structure of the world could change using subclasses, with no need to change the original class structure.

As was described in the past couple weeks the Game of Life I designed worked via two classes, God (the controller), Angel (a mutator agent class), and an abstract class World that maintains a record of the state of the world, mutated through an interface held by the angel class, and displayed by an access only interface.

To demonstrate arrays for the requirements and also to show the flexibility of the World abstract class, to really put the test the purpose of abstracting the access to the data structure, I made a version of the World class that holds the state of the world in dynamically allocated 2 dimensional arrays. It was reassuring implementing a new subclass of World and also using the interfaces for both subclasses in a new output environment. It aptly demonstrated the benefits of the abstraction and encapsulation.

I did find though that while this design method had the benefit of encapsulating the data

structure allowing me to use different subclasses interchangeable it also came with the drawbacks of top-down design. Primarily that it was difficult to add functionality to the program after it was developed because all the interface methods were defined before development began. It would be possible to add more functionality, but an interface change would often require changes to many different classes, adding the appropriate method to the interface and then implementing it in all implementations. Additionally adding functionality through subclass specialization was not an available option because the controllers acting upon the subclasses were doing so through pointers to the interface classes, meaning that they only had access to these defined functions (an example of the slicing problem).

The most exciting part of using the ncurses library was being able to receive non blocking input from the user, as in the program could appear to be continuously operating while still receiving input from the user. I originally used clock() and the TICKS_PER_SECOND macro defined constant to control the animation speed. But now it is controlled by the length of the delay for input before returning the ERR character.

Reflection

I entered this course with extensive programming familiarity but as this was the beginning of a new academic direction for me I made it my goal to fill in any knowledge gaps I had accumulated during my self guided studies. I had learned C++ many years before but had never used the object oriented features of it. And before this class the language I was the most familiar with was Java. C++ provided a great chance to get familiar with the more nitty gritty memory operations such as using pointers. Additionally it made me reflect on how to use the idioms and patterns I had previously learned in another environment. There is almost always a way to do the same design techniques, but learning a new syntax for them makes the underlying concepts of the design pattern come in to focus.

One of the more tricky patterns to replicate in C++ was the concept of an interface. It took a lot

of experimenting and white boarding to figure out how to use pointers to a class with pure virtual methods, and friend classes that are constructed and delivered by the implemented subclass. This comes at a bit of a cost to readability though. A class that has multiple of these psuedo-interfaces ends up with functions spread across multiple classes. It would have also been possible to have attempted to do this via multiple inheritance, but that would have its own messiness. I suppose I am saying that I look forward to using the implements tag again, but this exercise has helped me learn how to find common idioms across languages.

This class was the first time I used containers that were optimized for searching like map and set. The first time I used a set was on the dictionary class and I knew that for it to function well it would be prohibitively expensive to be searching it iteratively. This first step into these associative containers led me to always be thinking of better ways to organize my data. When design the game of life as a challenge to myself I tried to imagine a design that world work in an infinite world, for counting neighbors I decided upon using a map that would associate a coordinate (the key) to a count of neighbors (value) and it would only contain cells that bordered live cells. To achieve the same with a 2d array would be impossible over an infinite universe.

Pointers and exception handling were another area that I heavily explored this term. I had previously never known how to properly use the exception handling try-throw-catch form and when primarily using Java, a language running on a virtual machine, I had been able to avoid using pointers, and avoid having to learn how to prevent segmentation faults. Now when I read the small stack trace returned after a runtime error I'm able to spot the exception class that is thrown, better able to understand the cause of the error, and possible even wrap the erroneous code with a try and catch as a means to begin debugging. Becoming more familiar with pointers shed light on the real inner workings of arrays and helped me understand how to use iterators (another thing I had avoided).

Many of the other students in the class, and the professor, say they despise using GDB, but I have found it to be very helpful. Either for finding what line of code caused a runtime error or for

tracing the source of a logic error using the display feature to track the changing value of a variable as the program executes line by line. Another technique I used for debugging was of course logging, this began in the class with just very verbose cout statements mixed in with the programs normal output. Later as I learned more about streams, file IO, and Linux stream redirection I was able to come up with better ways to output debugging information. In this final project in which I was using the ncurses library for output it was very useful to use a file stream output to log the key presses and program response. I used a separate terminal running tail -f on the file to get live debugging feedback separate from the normal operation of the program.

Streams were one of the most powerful tools available in C++ that I at first underestimated and then later throughout the course tried to become more and more familiar with. I first used Linux stream redirection to automate the testing of simple functions, replacing the cin stream with a file input and outputting the results to another file. Later I learned how to write a function that receives the parent class istream and was able to operate on files, strings, and the regular cin stream.

Finally some of the most exciting learning this term has been getting a greater familiarity with using some of the more engineering oriented tools of computers. I have become more familiar with command line interfaces using linux/unix terminals and ssh. I now find myself using the terminal with cat and vim more often then gui file browsing and editing. The most exciting was learning to use the g++ compiler. My previous learning in development had always been tied into learning an IDE and if the “run” button stopped working for some reason I would be out of luck. Previously the biggest barrier to learning a new language was not learning the new syntax, but rather figuring out how to get something to compile. Now after installing and using g++ I am confident I could set up a compilation environment for anything.

Special Note on Requirement 1

after the initialization this program uses ncurses input, to see input using the cin and cout streams use the command line argument -s, this will allow you to input the terminal dimensions manually. This is the function called by the exception catch if the resolution isn't detected but to ensure that you can test the execution of this input function the -s command line argument is provided

Locations of Requirements

Requirement #01: demonstrate simple IO
- SwansonFunctions.cpp line:40
- cursedGameOfLife.cpp line:265

Requirement #02: demonstrate explicit typecasting
- SwansonFunctions.cpp line:76

Requirement #03: demonstrate logical operators ! && || ==
- SwansonFunctions.cpp line:108

Requirement #04: demonstrate a loop
- cursedGameOfLife.cpp line:235

Requirement #05: demonstrate a Random Number
- SwansonFunctions.cpp line:175
- GOL/WalkerMaker.hpp line:187

Requirement #06: demonstrate error categories
- GOL/arrayWorld.hpp Line:61
- GOL/arrayWorld.hpp Line:81
- GOL/MapSetWorld.hpp line:233

Requirement #07: demonstrate debugging trick
- GOL/arrayWorld.hpp Line:64
- GOL/MapSetWorld.hpp line:227
- cursedGameOfLife.cpp line:355

Requirement #08: demonstrate function and overloading
- SwansonFunctions.cpp line:59

Requirement #09: demonstrate Functional Decomposition
- cursedGameOfLife.cpp line:240

Requirement #10: Demonstrate scope
- cursedGameOfLife.cpp line:40
- cursedGameOfLife.cpp line:164

Requirement #11: Demonstrate Passing methods
- cursedGameOfLife.cpp line:136
- cursedGameOfLife.cpp line:315

GOL/MapSetWorld.hpp line:65

Requirement #12: demonstrate cstring and string
cursedGameOfLife.cpp line:48
GOL/WalkerMaker.hpp line:215

Requirement #13: demonstrate recursion
GOL/arrayWorld.hpp Line:55

Requirement #14: demonstrate Dynamic multi-dimensional arrays
GOL/arrayWorld.hpp Line:104
Requirement #15: demonstrate command line arguments
cursedGameOfLife.cpp line:96

Requirement #16. Demonstrates definition and use of struct
GOL/GameOfLife.hpp line:16
GOL/God.hpp line:18
GOL/MapSetWorld.hpp line:218
cursedGameOfLife.cpp line:205

Requirement #17: demonstrate pointers
GOL/arrayWorld.hpp Line:38
GOL/MapSetWorld.hpp line:136
GOL/MapSetWorld.hpp line:186

Requirement #18: demonstrate namespace
SwansonFunctions.hpp line:21

Requirement #19: demonstrate Header
SwansonFunctions.hpp line:12

Requirement #20: demonstrate makefile
see 'makefile'

Requirement #22: demonstrate overloaded operator
GOL/God.hpp line:50

Requirement #23: Demonstrate File IO
cursedGameOfLife.cpp line:361
cursedGameOfLife.cpp line:379

Requirement #24: demonstrate polymorphism and inheritance
cursedGameOfLife.cpp line:189
GOL/WalkerMaker.hpp line:116

Requirement #25: Demonstrate exception
cursedGameOfLife.cpp line:147