Reflections: Arrays and Classes

Spencer Moss

January 15th, 2016

1 Design

This project was my first foray into the language of C++, but not the OOP as a programming model. My previous experience with OOP lies within the bounds of Java and Python, but I have never before done such a large project.

My initial design included two classes and a main, and is well reflected in my current version; a Game class and its respective files, a Player class and its respective files, and a `main.cpp` with the main function within it. I admittedly probably spent too much time thinking about the Game, as it was explicitly mentioned in the system's specifications, and not enough about the Player class that I wanted to implement. I included Player as a class in anticipation of adding AI to the program, after the initial 3x3 board version was functional.

Unfortunately, a mixture of learning C++ style and practices on the fly and general logic errors ended up giving me tunnel vision on the Game class and the main function, and as a result a few helper functions found in my `main.cpp` file may be better refactored as parts of the Player class.

Elements of the Game class that were not originally drafted in my initial design are the `get_char_at` and `get_board_size` member functions, along with the `board_size` member variable. Initially, the `game_state_t` enumeration was present in a separate header file, `game_state.h`. The use of the enumeration did not turn out to be as important as I had first suspected for the program as a whole, and could probably be refactored out entirely.

While practically all of my initial thoughts for the design of the program remain intact, a large amount of the program was hurriedly added as I had realized that I had not thought enough about the actual process of playing the game. A lot of functionality remains in `main.cpp` that should go elsewhere, be it the Player or Game class, or perhaps another class entirely.

2 Testing

For the purposes of testing, the first thing that I set out to do was write a `test.cpp` file that would perform unit testing on individual methods of the two classes I wrote, Game and Player, before writing the classes themselves. I primitively, perhaps naively, opted to simply `cout` information and expected values from methods. This approach worked very well for many of the methods I implemented, but occasionally a bug would appear that would escape me. Only two bugs appear, but they are relatively minor in terms of impact, and are both related to user input and validation.

1. The first bug occurs during any y/n prompts. If you provide more than a single char, every single char is evaluated until a valid char is hit. Additionally, the program yells at you once for every invalid input.
2. The second bug occurs whenever user input is taken; if invalid input occurs, valid input will be met with no response. Two valid inputs in a row will continue normal program execution.

The unit testing slowly broke down halfway through the program, as the design did. Not enough thought was given to what to do with the Game and Player classes once they were completed, and all additional functionality was essentially brutally manually tested by my roommate trying his best to break everything. To date, no errors have been found besides the two listed.

One specific bug required me to use the GNU Debugger to solve, which was introduced when the board array became dynamic, and was the result of calls to `has_player_won` with coordinates outside the size of the array; this was one of many difficult problems encountered outside the calm test bed of the `test.cpp` unit tests.

3 Features

The program meets all the specifications of the system, in addition to a few other things. Every aspect of the game may be changed after initial execution during a replay, save for the size of the board; this includes AI and turn order. A secret 's' case may be entered when prompted for turn order that allows the 'AI' to play itself, inspired by a bug where the AI flags of the Player objects were not reset upon restarting the game.

4 Final Thoughts

Overall, the project was extremely stressful as I had to quickly learn quite a few things that I had never done before at all, or at least not in C++. Input validation is still a bit of a mystery to me; `cin`'s error bit and the way `.clear()` and `.ignore()` work aren't 100% clear to me yet, as I do not understand the bug where correct input is not immediately accepted. The project slowly deteriorated in design quality halfway through, in addition to becoming quite verbose, but as a learning exercise I found it to be quite invaluable. At the end of the day, all the desired functionality is there, with (perhaps overly) sanitized input and at least one additional feature.