Menu System

Right off the bat I decided I would use a third party library, as I knew I would likely end up using one anyway and trying to cram one in near the end for the boost task would’ve been very hard to do. As a result, I spent the first four days wrestling with all the features it introduced to me. The library I went for is called PDCurses, a version of NCurses designed for windows (all features work between both libraries as they are named the same thing (clear() to clear the screen etc.)).

As a result, I ended with a menu system that I was pleased with, but soon realised that I would need to create multiple menus that looked very similar, so I moved as much functionality as I could over to a master menu class that I could then inherit from. Once that was done, all I simply needed to do was pass a vector of strings (containing the options on the menu) to the master class and the class would do everything for me.

After implementing this however, I discovered I was in desperate need of a state manager, so I created one with which to move from one menu to another with. Each menu (and game, once over or the player left) would return a string which would correspond with an enum known as states, causing the switch statement in the main loop to swap to a new menu or game (or exit the program).

Blackjack

Blackjack was the hardest by far for me simply due to the amount of rules and exceptions that are present in the game. Splitting the hand proved to be particularly tough, however I ‘cheated’ it by simply storing the duplicate card as a lone string, then playing the game as normal. When the player would’ve lost, I threw out their hand and added the string back in as a new hand. In short, rather than having two hands, the player has one hand and a separate card.

Doubling down proved difficult too, and the end implementation of it feels simply like drawing one more card and then standing. The code towards the end of the cpp file looks very sloppy, and the update loop and checking for winning, losing or drawing is simply hacked together. Overall, this is my weakest of the four sections.

Hangman

Hangman proved to be quite easy once I had learned how to read and load from files. When it came to actually drawing the man as he was being hanged, PDCurses proved to be invaluable, as it allowed me to print at coordinates instead of having to add new lines and spaces. This allowed me to get it done in no time at all, though I’d have preferred to cut the amount of if statements used for it down somewhat. Perhaps loading the man from a text file would’ve cut down on the unnecessary code.

Snake

Once again, PDCurses proved invaluable when it came to making snake, as all I had to do was input the coordinates of the snake’s head and body (stored in two vectors, one for the Y coordinate, and one for the X coordinate). An enum was used to store the direction of the snake’s head, that way I could move the snake in the correct direction and even print different characters depending on the direction it was facing (‘<’, ‘^’, ‘>’, ’V’).

On top of this, PDCurses had a built in method of not having to wait for the users input before carrying on with the program, through the use of nodelay(). Thanks to that, the snake would move on its own without the player having to mash a direction to get it to move. I did however have to use chrono, as otherwise the snake would’ve ran head first into the wall and died the moment the game was loaded up (as the program would execute everything and hit the game over screen immediately) (I did also have a good hour where I forgot to turn nodelay() off, so all three games played at light speed).