Fruit Machine - Report

When I first started working in this project, the first thing I thought it would be more challenging was the fact that each column of the fruit machine had to be rotating and the user gets to stop each column individually.

However, after some online research and some ugly attempts, I found a library called ncurses which basically provides an API that allows the programmer to have more control over the terminal interface (Windows Command Prompt). Unfortunately, this library was written for UNIX-like systems.
Luckily, there is a library similar to ncurses that was written for Windows applications called PDCurses.

With this library I was able to print anything to the console window exactly where I needed, using a function that takes the position (the column and the line) and the string or character to be printed.
Also, this library provides a function to clear specific parts of the console window, which makes it really easy to solve that initial challenge aforementioned and other difficulties that I encountered later like the console screen flickering. See below the functions of this library that I used.

The solution I found for the slot machine columns changed a bit during the development of my project. Firstly, I decided to use an array of chars, so that every char represented a fruit (B – banana, A – apple, etc), but I thought that was a bit too simple so I decided to improve. Secondly, I decided to use an array of strings, where I stored the whole words of the fruits (firstCol[7] = {“Banana”, “Apple”, etc}), but then I was having problems passing each column (array of strings) to the function that locks each column. Lastly, I decided to use vectors of strings, which are much easier to manipulate (pop_back() and push_back() functions) and with this solution I had no problem passing the columns (vectors of strings) to functions.

When we first run the program, the first menu shows us the different game modes and an option to check the game rules and the prizes.

Rules:

You start with 100 credits and each game costs you 15 credits.
You can play the Normal Mode, the Fast Mode and the Ultra-Fast Mode:

- Normal Mode(1) requires full user interaction like a normal slot machine.
- Fast Mode(2) requires less user interaction but you can still control when each column stops. You also get to choose every 10 games if you want to keep playing.
- Ultra-Fast Mode(3) requires no user interaction and the game will run until you have less than 15 credits. You also get to choose every 100 games if you want to keep playing.

Prizes:

Each DIAMOND - 50pts

DIAMOND DIAMOND DIAMOND - 1000pts
BANANA BANANA BANANA - 150pts
DIAMOND DIAMOND BANANA - 110pts
BANANA BANANA DIAMOND - 60pts
BANANA APPLES DIAMOND - 50pts
BANANA BANANA APPLES - 10pts
BANANA APPLES ORANGE - 0pts

Some relevant functions of PDCurses:

- move(lin, col) – move cursor to position (lin,col)
- mvaddstr(lin, col, <string>) – print <string> on position (lin,col)
- mvprintw(lin, col, <string>, variable) – same as mvaddstr() but with the ability to print variables. Eg: mvprintw(lin,col, "Credit = %d", credit)
- refresh() – applies changes to screen. Needed every time we want to add or clear something on the screen
- clear() – clears the entire console screen
- clrtoeol() – clears current line
- initscr() – initialises PDCurses
- noecho() – don't print the character entered to the screen. Useful to catch user option without messing with the screen
- cbreak() - don't wait for user to press ENTER after pressing a key
- curs_set(X) – if X=0 => hide the cursor. If X=1 => show the cursor
- getch() – catches user input. Useful for loops where we don't want the program to stop when waiting for user input. See nodelay() below.
- nodelay(stdscr, TRUE) - causes getch() to be a non-blocking call. If no input is given, getch() returns ERR and the loop keeps running
- endwin() – exits PDCurses

Notes:

- I decided to declare the variable credit as global because this variable is used and modified by many functions and this way, I don't have to pass it as an argument every time.
- I've used the *map* container to store the number of occurrences of a certain result, like Jackpots, two symbols, Diamonds, the number of games played and the cash-flow (credits spent/earned). I declared the *map* stats as global because its values are used many times throughout my code and by many functions. Therefore, I'm avoiding having to pass the *map* as an argument every time.
- I used *constexpr* instead of *const* to declare constant variables like the speed of the slot machine and the price because I read that it helps the program to run faster. MicrosoftDocs
- I included *chrono* and *thread* libraries to use a function like sleep(n_seconds) to control how fast the columns overprint.
- Even though I know it's not best practice to use infinite loops (while(true)), I decided that was the best and the "cleanest" implementation for my solution.

Example of call trace (core functions):

Normal Mode:

```
main()
    loopGame()
        playGame()
            slotMachine()
                lockFirstCol()
                lockSecondCol()
                lockThirdCol()
                    updateCredits()
            evalResult()
        displayStats()
        cashOut()
    exitGame()
```

Ultra-Fast Mode:

```
main()
    loopGame()
        playGame()
            ultraFastMode()
                updateCredits()
        displayStats()
        cashOut()
    exitGame()
```