Project Class Design

Justification

Note: I have changed my design significantly from Assignment 2.

The `Player` class contains basic mutator functions and a generic post-game function that mutates multiple related variables (e.g. wins and win streak). The logic in the added function would be consistent across other games, meaning that the driver file wouldn't have to do it directly. I also moved the skill level variable to this class as it is likely to be used to affect decisions made by a sophisticated computer player if implemented. Otherwise, this class's member variables include the name, total score for a program session, games played, games won, and a win streak.

I've made the `Board` class faithful to the assignment brief (rows, columns), but also included a 2D vector representing the board pieces as integers (or characters as they are related). This vector can be extended upon calling a 'reset' function (which is only called during post-game). This is to ease the integration of the skill level feature.

In terms of the application, I shared its functions across the driver file (commands and interface), and an inherited Board class called '`CodeBoard`' (code generation and comparing guesses). This helped separate presentation from the game logic.

The `SkillModifiers` header file contains an array of characters that `CodeBoard` can choose to create the code from, and an array containing set number of integers representing the number of rows for the board based on skill level.

Object-Oriented Design

I ensured that the `iostream` library is only used in the application file as it would be handling the player's commands and output. Instead of declaring whole namespaces, I've only declared required operations into the scope (e.g. `cin` and `cout`) to reduce conflicts with future versions of C++.

Theoretically, the `Board`, `CodeBoard`, and `Player` class can be ported to another project (such as a graphical version of Master Mind) with minor adaptations.

Project Issues

First Issue

Throughout using my original plan, I encountered a major issue with the design in which the player and board objects initially set wouldn't carry over between `gameOver()`, `playGame()` (renamed to `playMasterMind()`), and `resetGame()`.

To solve this, I had to create a nested loop function which runs for the entirety of the game section (called by `playMasterMind()`) that takes in Player and CodeBoard objects using pointer parameters.

These pointers were initially made by `main()` and would be assigned to an initialised class by a new function called `setMasterMind()` which uses double pointer parameters to mutate the `main()` pointers.

Second Issue

I also realised that some of my driver functions were doing more than they were required to do. For example, `gameOver()` calculated the game score, gave feedback, asked to play again and checked for promotions.

I had to separate my driver functions that would be called by the game loop function. For the same example, `gameOver()`'s original purpose is split across itself, `getScore()`,`determineComplexity()` and `determineSkill()`.

Future Improvements

If I were to do this project again, I would have made the game application its own class, where it would manage the board and player objects as private pointers and request inputs from the driver file/user when appropriate.

This change would have made the project better resemble the object-oriented design used in standard video games, where their Game objects manage other classes such as a players, stages, and enemy factories.

I would also consider splitting the driver file's command reading and menu generation into one or two dedicated classes.

This change would have reduced code mass of the driver file, making it easier to read. In addition, it would also allow the project to be reused without requiring huge changes.

Another change could be the use of a '`Theme`'/'`Text`' class that can take text files containing feedback. This would allow for the ease of changing the game's theme (e.g. restaurant themed MasterMind).