Project Description:

The popular card game, Solitaire is now presented to you by our team, *Byte me*. The goal behind this game that you can find on almost every computer or laptop using the windows operating system is to create four piles of cards - one per symbol- in ascending order, starting from the Ace (1) and ending with the King (13). These are called the foundation piles. The next step is to start building the layout, by putting down the first card face up and lay six cards face down next to it. Then, put one card face up on top (but lowered slightly) of the first face down card, then put a face down card on top of the other five cards. Continue doing this, so that each pile has one face up card on top and so that the left pile has one card, the next has two, then three, four, five, six, and finally seven.These piles of cards will become your tableau columns as you play the card game. The remaining cards are put separately so you can draw a card from them when needed. Four spaces are left in the top of the screen, a space for each foundation pile for the four different symbols. Now we can start out gameplay!

If there are any aces, place them in the space left for the foundation pile equivalent to that symbol. If there are no aces, start looking at the face up cards left in your tableau columns. You can only move a card if there is another card having a different color while also being higher by one digit than the card you moved. For example: if you have a six of hearts, you can either place a five of spades or a five of clubs on top of it. Keep placing the cards on top of each other until you cannot move anymore, then you can draw one card from the pile of hidden cards until you find a card that can be moved to our tableau columns. If you moved a card (or even multiple cards) from one of the tableau columns, then the card that was behind it must be visible so you can move that new visible card if possible. If one of the tableau columns becomes empty, you can only start it with a ***K***. Then you continue playing until you win, or until you get bored after not finding a valid move (which is the case most of the time!)

\*\*\*\*\*

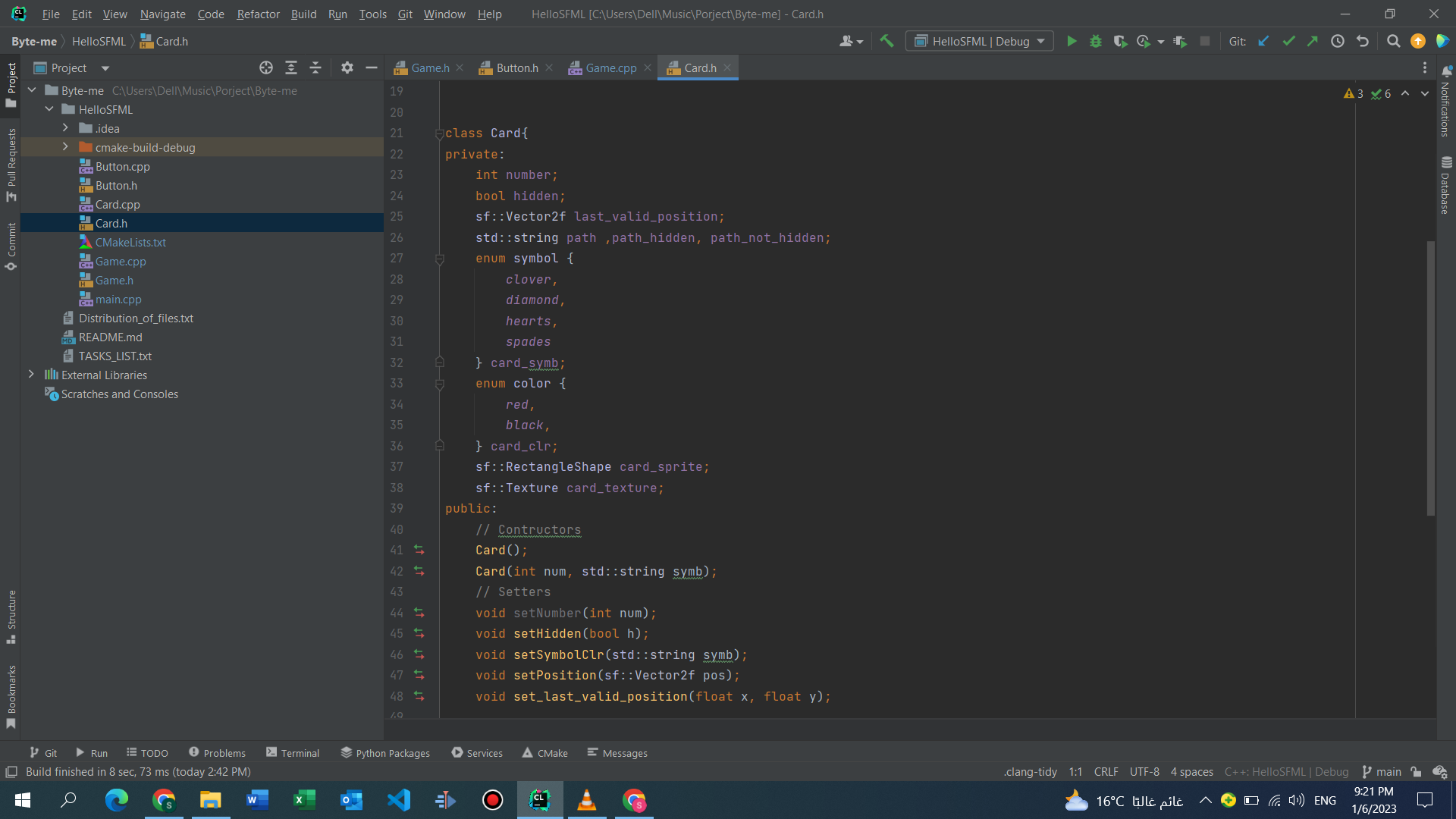
Program Structure:

***Card*** class:

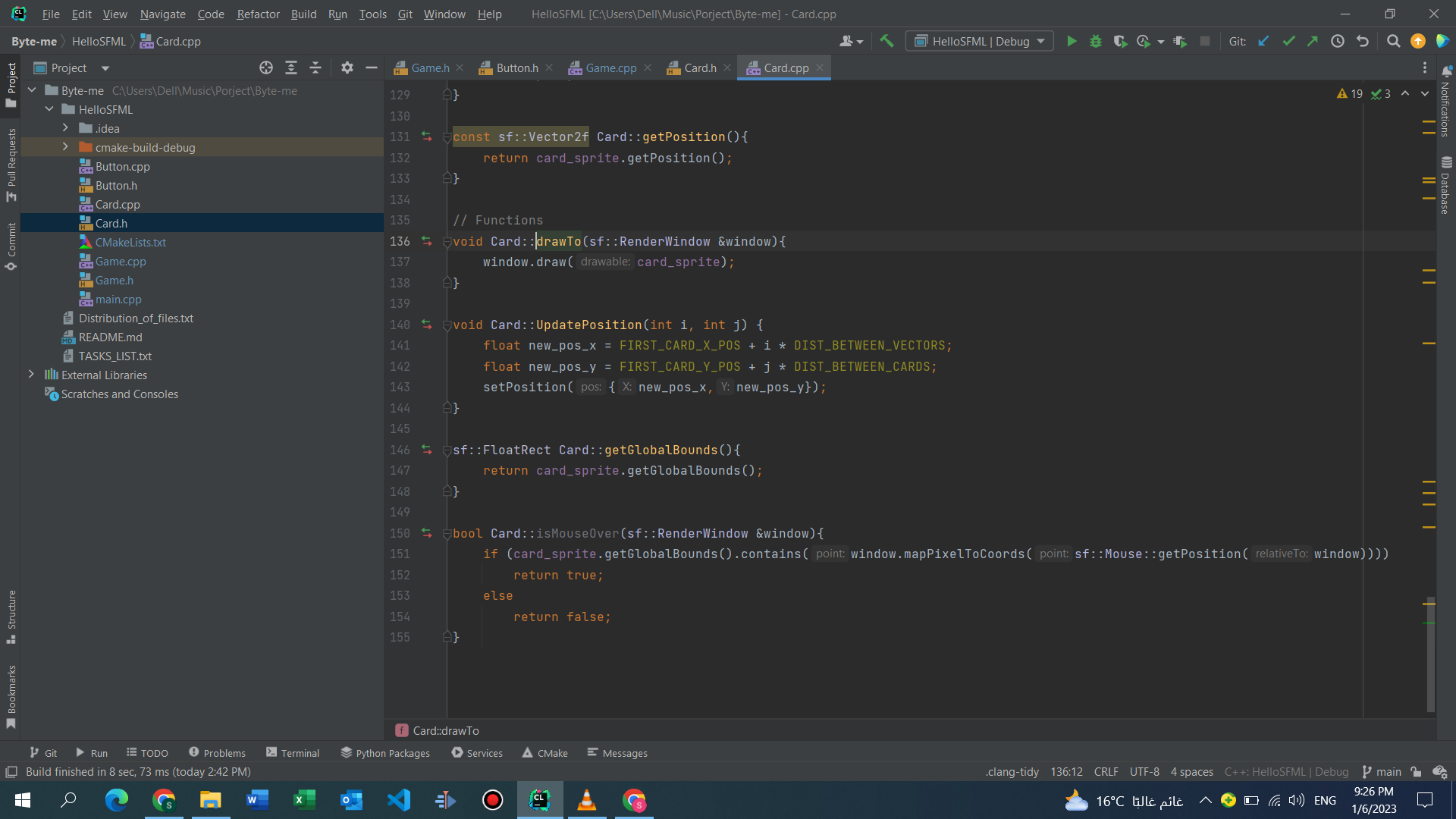
This class is responsible for storing all the variables, GUI related addons, functions of each card in our game, like it’s number, symbol, position, whether it’s hidden or not, etc..

We decided to use *enum* for the card color and symbol because it’s similar to Boolean variables, but instead of 0 or 1, it has a similar functionality where you determine the different set choices you can choose from, like red or black.

The functions included are either setters and getters for our variables, of GUI related functions, like the *drawto function*.



The class in code

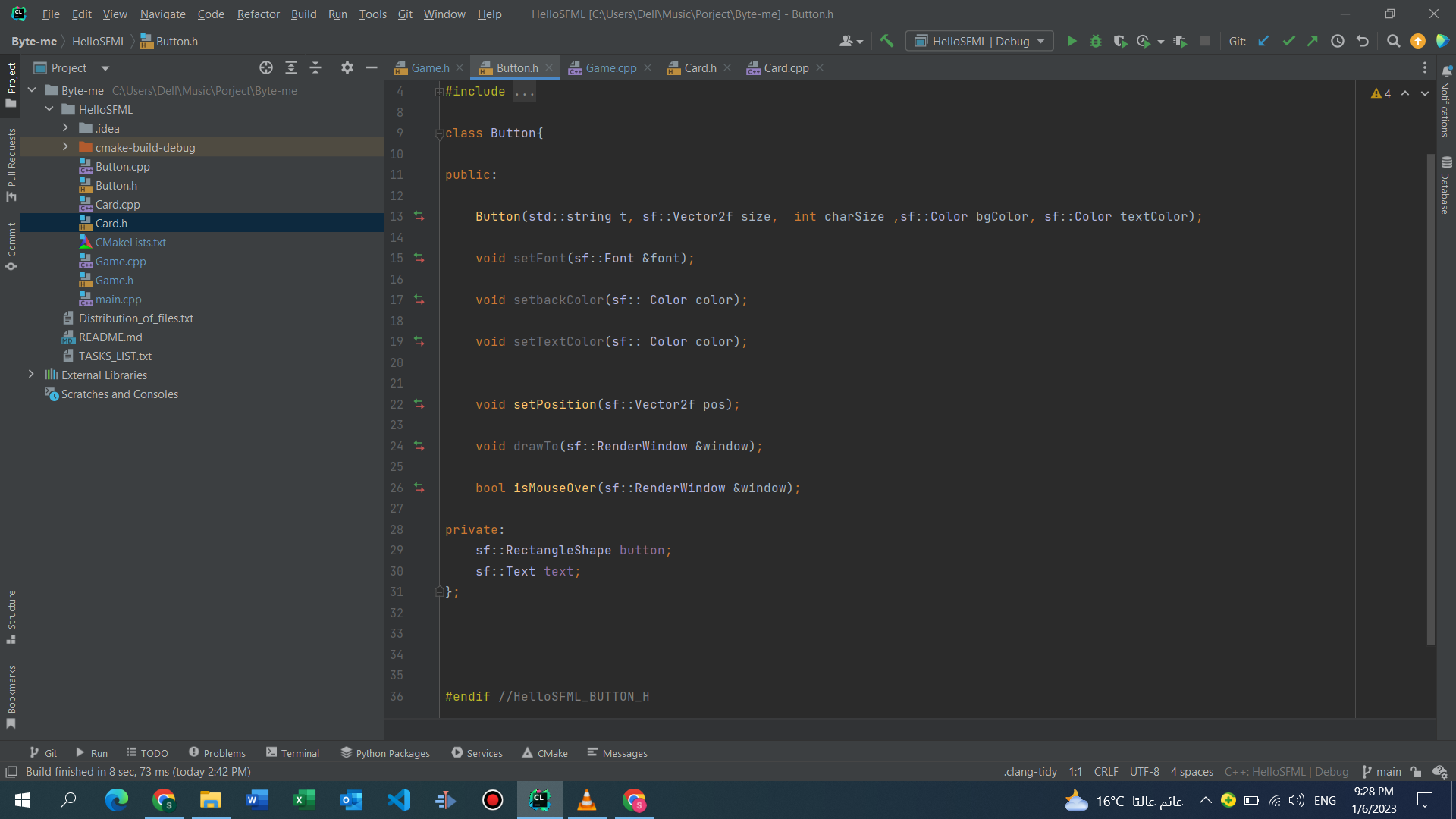


GUI Realted functions

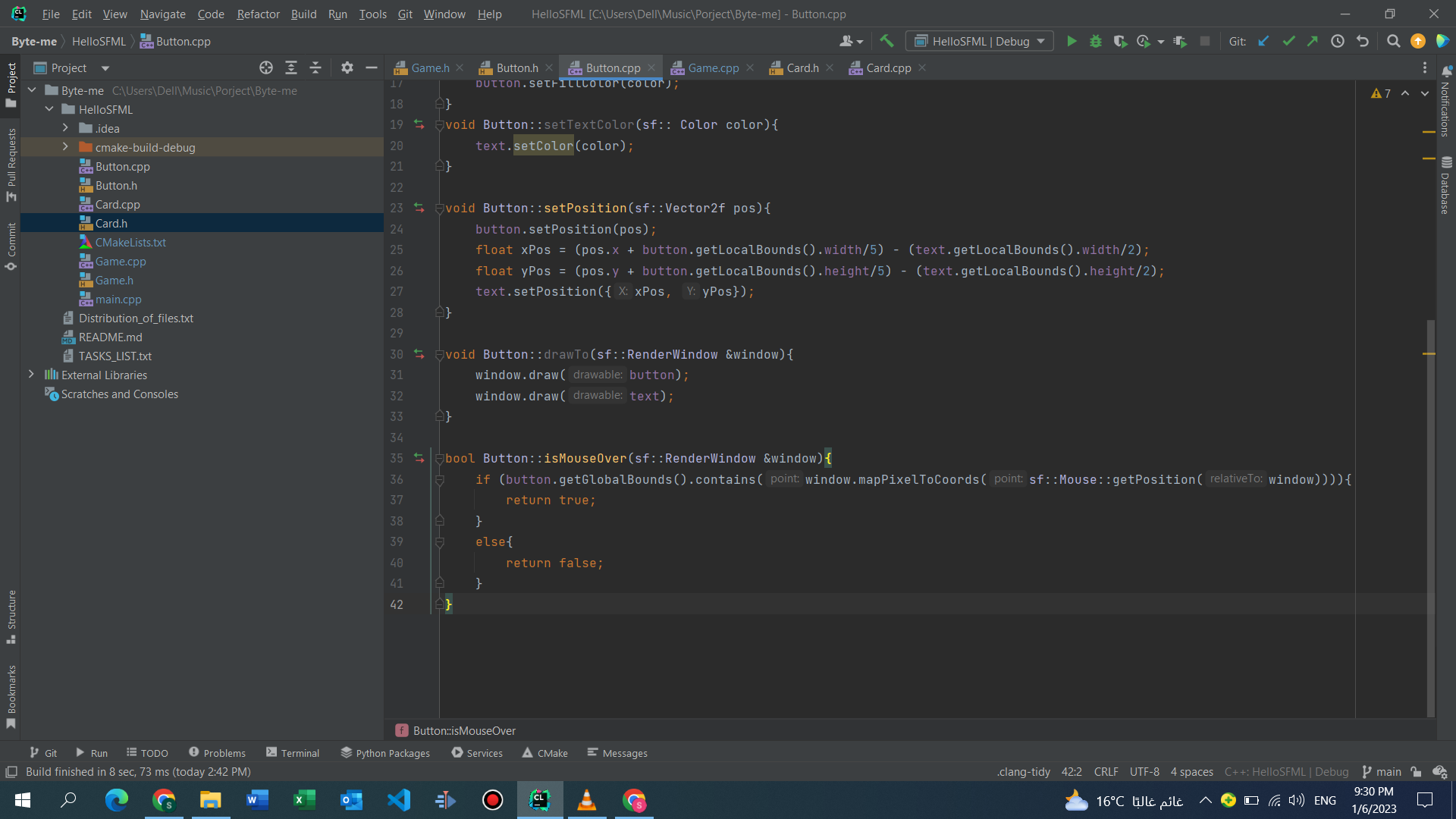
\*\*\*\*\*

***Button*** class:

This class is responsible for storing all the variables, GUI related addons, functions of any button in our game, like it’s position, whether the mouse was over it or not, etc..



The class in code



The isMouseOver function

\*\*\*\*\*

***Game*** class:

The most important class in our game, where all the back-end and front-end code is written, including all the containers used to store the cards, the different buttons used throughout the game, the Boolean variables used as flags inside the different functions of our class, and all the GUI related variables used in displaying the game to the user. The function are split into private and public functions, the private functions are the functions called in the constructor the moment the game is started, and these functions mostly focus on making the game ready, like shuffling the cards at the start of the game, then displaying the game to the user so he can start playing, including the background, the buttons, the score, the timer and of course, our cards. The public functions are split into 7 categories:

1. Constructors and destructors:

a. Mainly used in calling the private function at the start of the game and freeing the space that we used throughout the game when we reach the end.

2. SFML related functions

a. These functions are responsible for checking on any event related to the GUI aspect of our game, like checking if the game is running, the mouse clicks, moving the cards by dragging and dropping them, etc…

3. Move to and from functions:

a. These functions are responsible for checking if any card we click can be moved to any possible location in our game, and then moving that card (or cards) if it was possible.

4. Additional functions:

a. *Hint* function is responsible for printing hints when clicking the hint button to tell the user if there are any possible cards then he could click to move a step closer to winning, however; it was designed in a way to help the user most of the time while still allowing the user to think by himself from time to time

*b.* *Undo* function: If you do a play you regret you can just press undo so that it never happened!

5. Draw function:

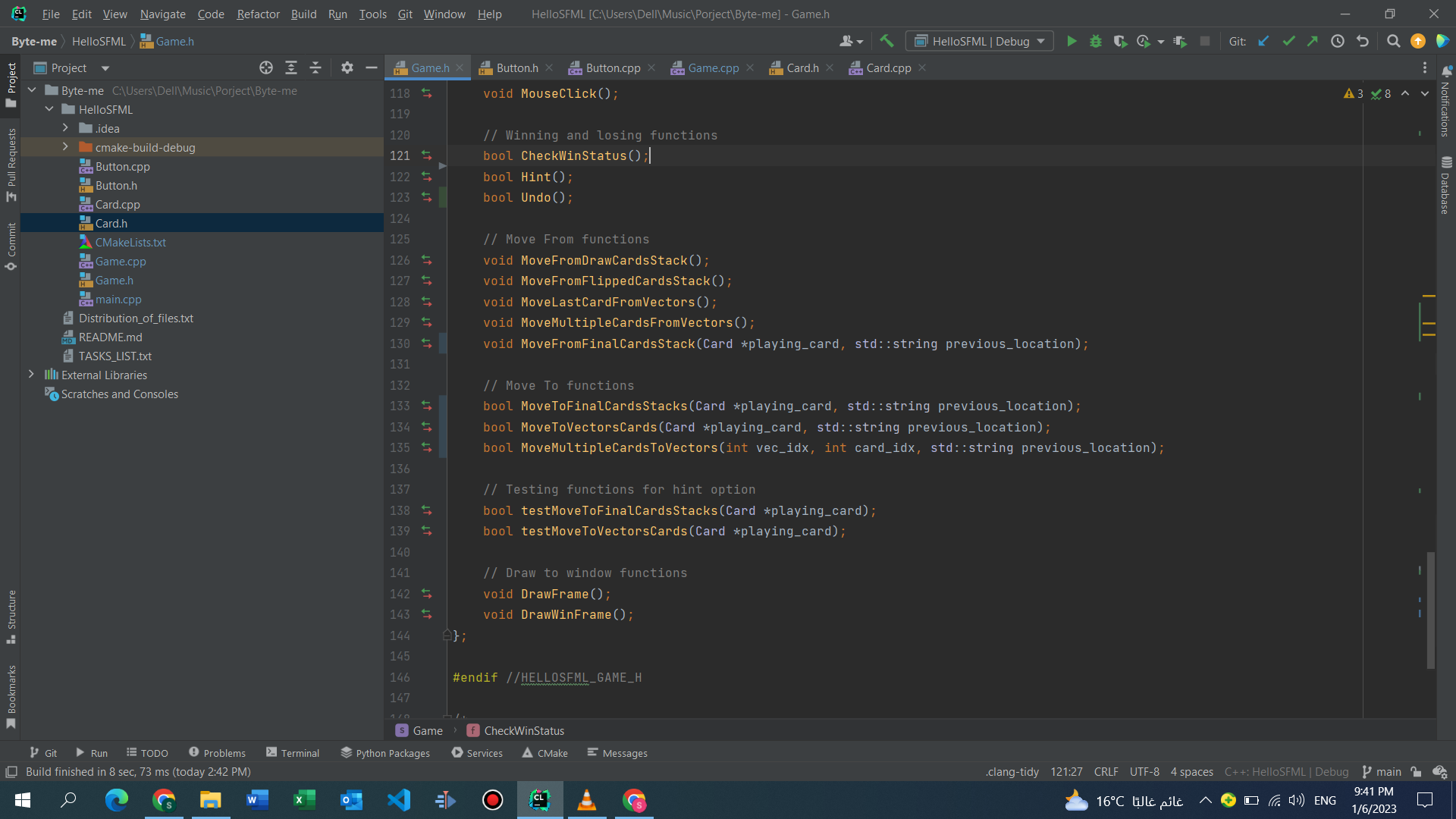
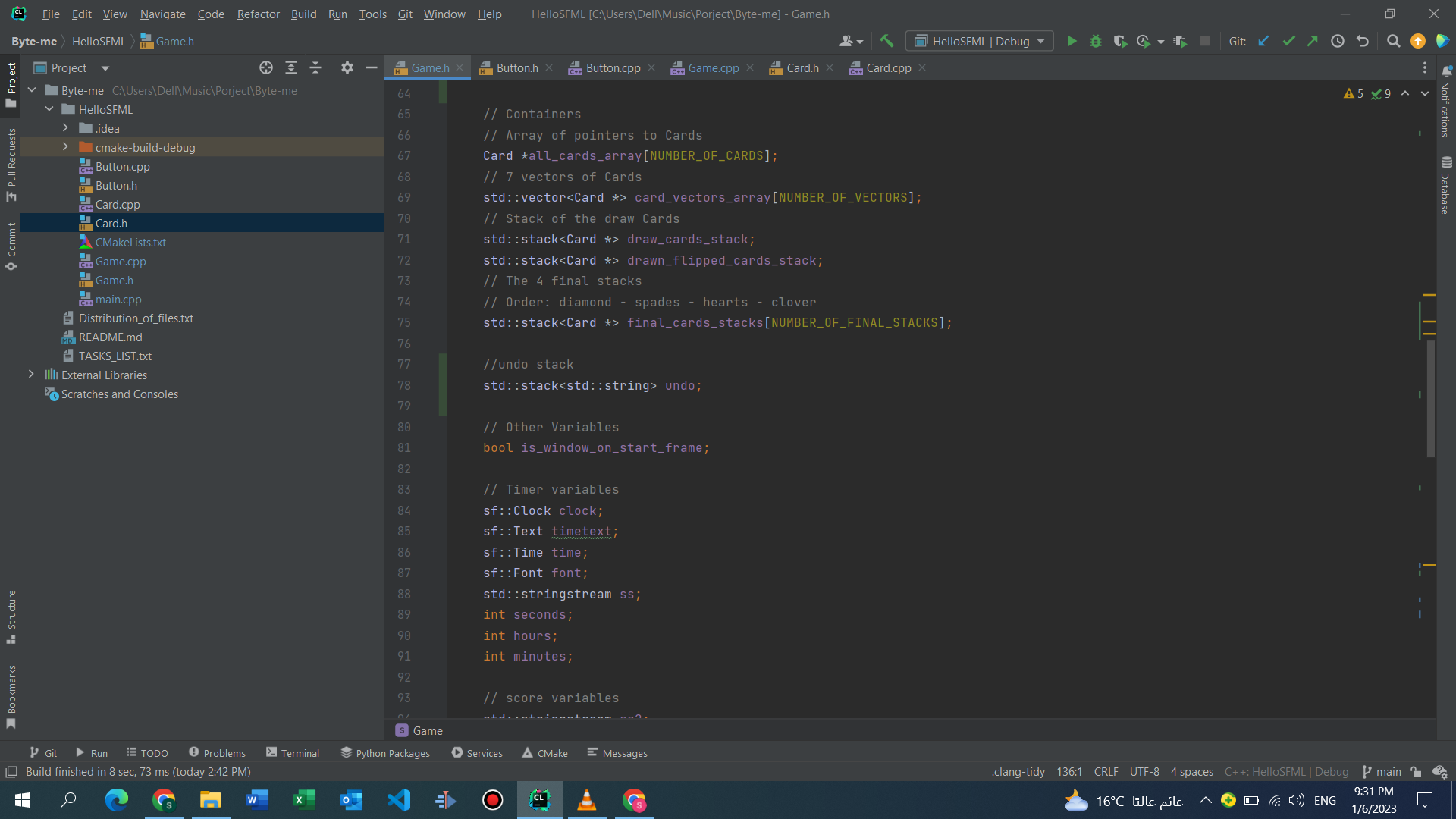
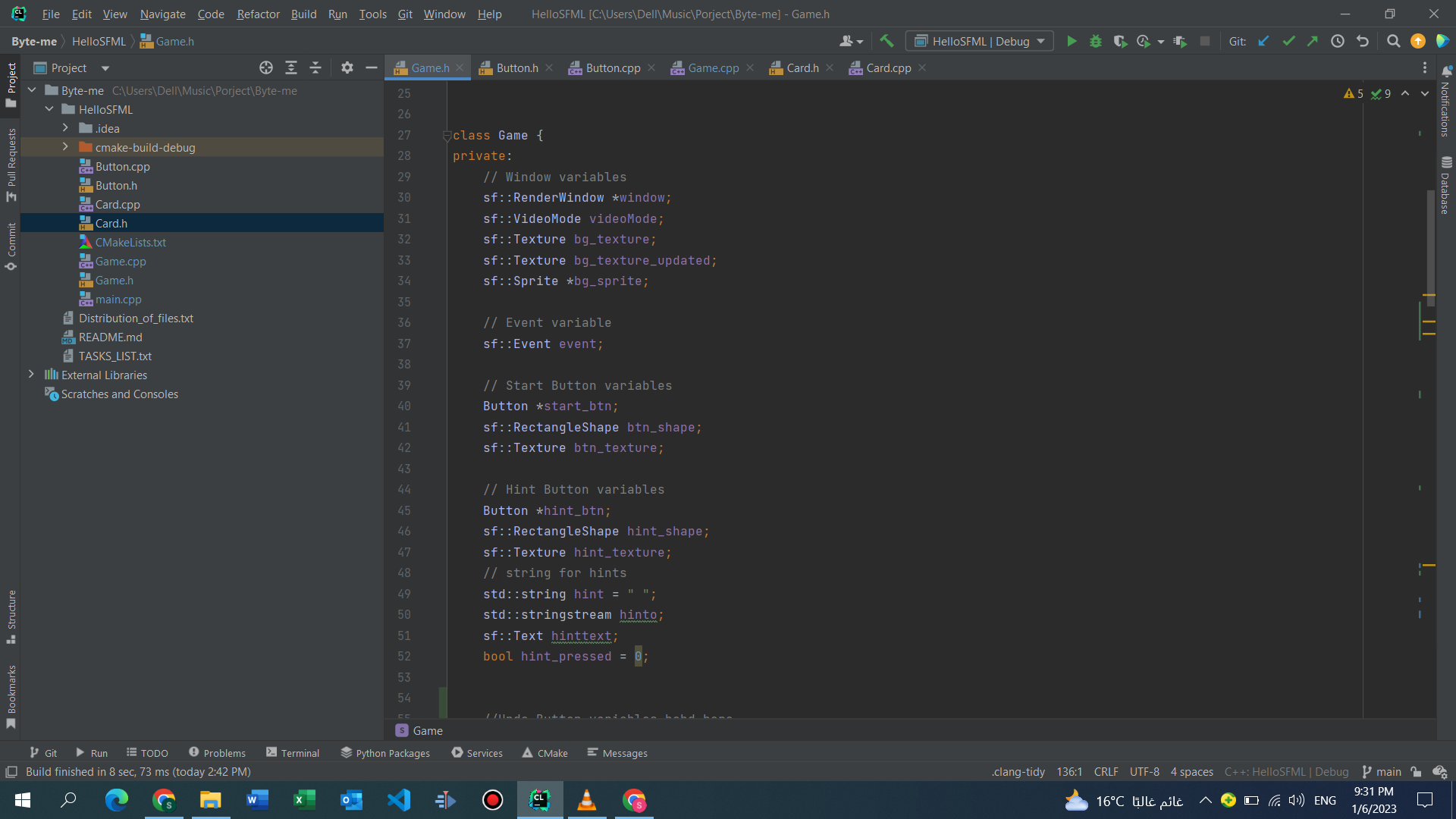
a. These functions are as the name suggest responsible for displaying the game to the user, and if he wins a special surprise awaits him!

6. Testing functions:

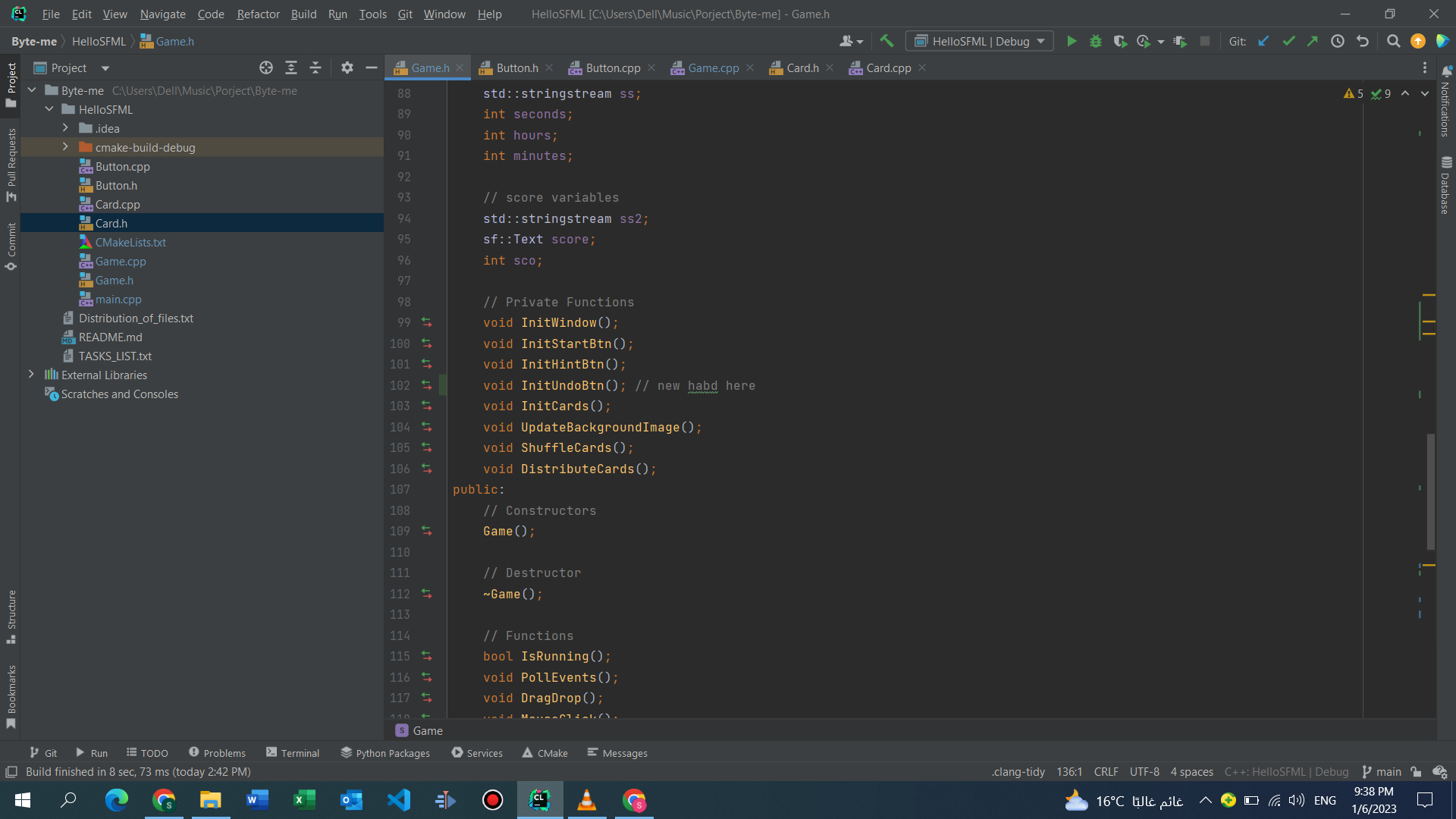
a. These two functions are only used in the Hint function to check if we can move a cards without actually moving it if possible

7. The *CheckWinStatus* function simply sees if you won or not

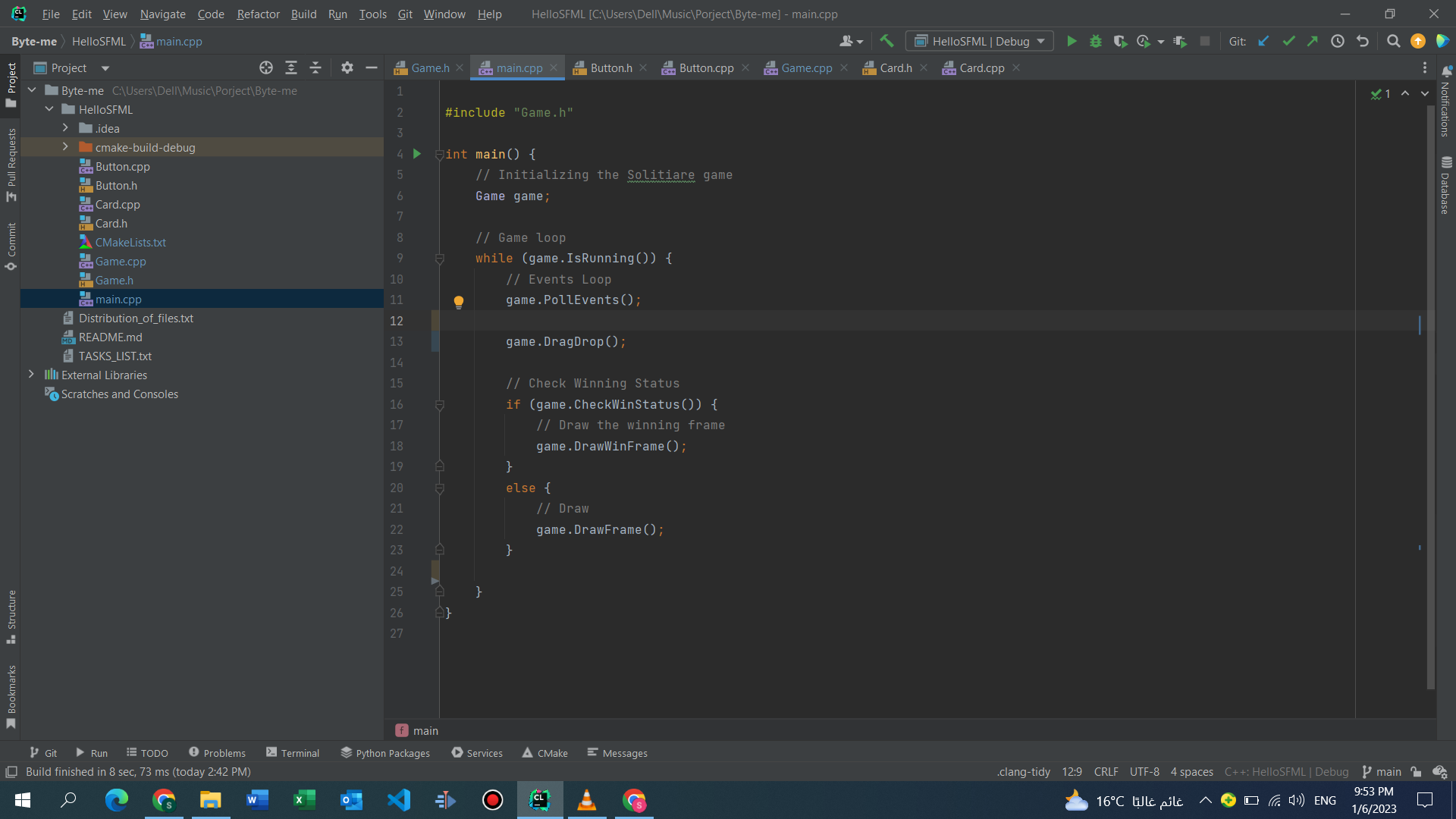
Finally, we have our main.cpp file which simply calls the game constructor to make our game ready, then while the game is running we check for any events, like; clicking, closing the game, dragging a card by mouse. Then it checks if the user win to show him the surprise, else it just displays the game normally



Public Functions



Private Functions



main.cpp

\*\*\*\*\*

Responsibilities:

Ayman Talat: Writing most of the back-end code and the mastermind behind the game to code logic, implementing the ***Game*** class, writing many of the different functions responsible for the game play, then organizing them inside the ***Game*** class, besides helping from time to time with the front end.

Mahmoud Khaled: Overall design of the game, the background, the design of the buttons, the cards, the icon, etc.. and implementing the functions responsible of moving one card by pressing on the card and moving it (Drag and Drop)

Sajed Samer: Implementing the ***Card*** Class and the ***Button*** class, linking the ***Button*** class with the button images, implementing the undo, hint, and move from final stack functions, assisted in move multiple cards functions, and helped in the logic behind the code of the timer, score, and some of the functions of the ***Game*** class. Last but not least, writing the report you are reading right now.

Yusuf Tamer: Writing the code behind the timer and the score, besides adding an icon to the exe file

Challenges:

1. Learning GUI: At the beginning we had no idea how to link GUI to our c++ code, which GUI LIBRARY to use, or how to even start learning GUI, but after we asked the other two teams on how they started as we started a little late, we decided to use the *SFML* GUI library to implement our GUI. It was hard to understand the syntax of this new library, and it felt like we were learning a new coding language, we were mostly just copy pasting from the internet, but we watch a couple of YouTube tutorials and understood most of the basics behind the *SFML* library and started using our new knowledge in writing the code and linking it to the design we had in mind for our game.

2. Implementing Class Game: After finishing phase 1, most of the code was written in the main.cpp file, making it look far from what we can call *clean code*, so we decided to put all of the functions, variables, and SFML related addons in one class called ***Game***. Ayman, who was the one behind the accomplishment of that mission had no idea when he started on how to move the SFML related addons to another file without ruining the code, or how to put all of our function in a single file without one function being un-accessible from the main, so he headed to the place where all programmers go to when they have no idea what to do, YouTube, and learned how to create the ***Game*** class.

3. Pressure: One of our teammates was terribly ill for a long time before the project so he had a lot of deadlines, assignments and quizzes left to do when we started the project, so the team was only three people most of the time, and we also had a lot of deadlines to be met, besides we were struggling with the GUI implementation, which put us under immense pressure for the first week, but after we finished most of our assignments, and properly learned about *SFML* the pressure started to lessen thanks to the great teamwork between the four of us.

4. Having Multiple versions of the project: Since the work was delegated between our team, each person added or removed parts from the code, while the others knew nothing of his edits, and when anyone wanted to upload his finished task on git hub where we stored our project we had to search for the parts he edited in order to add them line by line to our code; in order for the new lines to be matching with the lines added by someone else, which was a huge headache, as sometimes only one line was added in one of the many functions to be forgotten and found by luck when one of us was reviewing the code. We solved this problem by using the *git* commands so we can make sure that any edits will be added without the need of us to traverse the code line by line to see what was newly added by one of our teammates.