Text

Description automatically generated

School of Computing and Mathematics

SOFT336SL

# Cross Platform Application Development in C++

BSc (Hons) Software Engineering

G.P.C. Hettiarachchi - 10707218

Target Sum QT Application

2020/2021

# User Manual

## Installation

Target Sum QTApplication can be developed and deployed without the need for other dependencies to be installed. It just uses Qt's core principles. As a result, the application may be started simply by extracting the source directory and loading the project (.pro) file in Qt creator before performing the build/execute.

This application's source code is available in a Git repository I've set up: <https://github.com/Pubzzz/TargetSumQtApplication>

## Possible Errors

The only error noticed is "'random shuffle': is not a member of'std'," which is thought to have occurred as a result of std::random shuffle being deprecated in C++14. In this case, you may replace random shuffle with shuffle. Because std::shuffle is derived from algorithm, adding #include< algorithm> to your code should solve your problem.

# User Documentation

## Description of the program

## The software is a modest game that allows you to put your adding skills into action. It's time to sum up faster in order to win, as simple as that, once you've successfully extracted, built, and executed the application. The game is divided into three primary levels, which you can choose between what you find comfortable for you. The main header displays the level you are in, followed by a target value that you must total up to using the button options below that revealing alternative numbers that could sum up to that target.

You are given 30 seconds to figure out the correct sum; if you figure it out before that time, you win the game; if you get the sum wrong, you lose the game; otherwise, after the timer runs out, the game is finished. But don't worry, you may give yourself a second chance by using the "Play Again" button, which will give you a new target value to total up along with a fresh timer. Also, keep in mind that as you progress to a higher level, the target increases in value, making it more difficult to figure out, especially because the timing duration is capped at 20 seconds and the number of options you need to choose from also increases.

## Supported Platforms

Target Sum QT Application has been tested and validated to work on both Windows 7 and Mac OS at this time. On both platforms, screenshots of a game at various states and levels are shown below. While it is assumed that the software will run without issue on Linux-based platforms due to the program's sole use of fundamental Qt ideas, although this has not yet been confirmed.

Both verifications were carried out with the QT Creator 5.0 community edition, and the possible problems seen during this process were documented in the user manual, which should be kept in mind.

Windows 11 Version 21H2

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generatedTable

Description automatically generated with low confidence

MacOS Monterey Version 12.1

Graphical user interface, calendar

Description automatically generated

The Mac OS implementation resulted in serious User Interface inconsistencies, such as the clicked Styles of buttons being compromised with the default program backdrop, and the resizing properties not being applied in conjunction with the software package's menu bar. Despite the fact that the logic and functionality are as expected, and the software runs smoothly on the platform, I've attached a screen recording that you may use to show how the TargetSum QT Application works on the MacOS platform.

[](Screen%20Recording.mp4)

# Technical Documentation

## Description of the GUI design

Graphical user interface, text

Description automatically generated

As you can see on the screenshot, it contains all of the QObjects, Spacers and Layouts that were utilized to create the Graphical User Interface.

Graphical user interface

Description automatically generatedThe Pushbuttons were utilized to allow the user to select value alternatives for calculating the target; these buttons provide the game's core functionality, as well as a large backend functionality to suit various game stages, which I will explain in the code documentation. Another important component is the Labels, which vary their content depending on the game's state, letting the player know what's going on and what he may do next by providing instructions and information across the application. Not only that but also, the QActions were critical in allowing us to deliver a multi-level experience to the client by modifying the User Interface as coded as the action was triggered; I'll go into more depth later in this documentation.

## Documentation of the code

Text

Description automatically generated with medium confidence

Because the application was created using only one QWindow, all of the coding is contained in a single.cpp file. This was tested using numerous QT Designer form classes, which resulted in an error while instantiating objects on those classes while the application was running. Additionally, such solution resulted in redundant code, making the code difficult to maintain and alter. That is the primary reason for using a single QWindow, and the source code is available in the repository's test branches if you need to investigate the error, I experienced with using several form classes.

Furthermore, when it comes to the resources provided to the app, images were mostly employed to give the program an appealing icon, as well as icons for actions, to help the player readily recognize the levels by using a colourful collection of image resources.

Moving on to explaining the main structure of the code, the header file mostly contains private slots for certain signals and public slots for various logical uses, as well as three libraries, one of which is used to instantiate the UI window and the other two of which are used to function the timer. The.cpp file, on the other hand, uses the instance created in the header file to resize the UI window and display upon execution.

And explaining the subject of the primary functional source code, it also includes a special library file named "QRandomGenerator" that is used to produce random numbers, as well as a set of global variables making it possible to access them from throughout the code. I've also specified a CSS code line as a global variable so that applying the same style to the pushbuttons is simple rather than having to repeat the code everywhere. Not only that, but you can also see some of the other strategies I've used to eliminate redundancy, which I'll go through in more detail later. Allow me to discuss the essential function methods in depth, while you are welcome to refer to the inline commenting done throughout to acknowledge why and what has occurred.

Figure 1 – Initial code lines implemented from line 17 – line 36

The initial code lines to be executed after the program is executed without any user involvement are shown in the diagram below. As a result, the source code is written to display the Beginner level, which is the game's default level, and then call the function method to produce a random number between the level specific range and display it to the user interface, as well as change the timer label to 30 seconds. The timer object formed by the QTimer instantiated in the header file is then used to build a Signal and Slot to connect the timer to mytimer private slot so it may be updated with the time. Furthermore, the timer is set to 1000 milliseconds so that it will be updated every second, and the timer is also set to 30 seconds.

Text

Description automatically generated

Figure 2 – on Beginner action triggered from line 43 – line 66

An enum is used to keep track of the game's level, so when the player selects Beginner from the select level menu, it sets the level variable to "1" and ends the timer, as well as setting the counter value to zero, which you'll see when I explain the mytimer function. Continuing, the level label is set to "Beginner level" to indicate to the user which level he is on, and the timer label is set to "00:30" so that the player knows he only has 30 seconds, despite the fact that the text value of this label changes every second with the implementation of the timer upon execution, which will happen when the mytimer() function is called. The myRandomNumber() function is also called, which sets the target label and push button values, enables, and resets the buttons to their starting colors using particular functions, and starts the computation when the calculation() function is invoked. (Further details on these function techniques can be found in the following sections.)

Text

Description automatically generated

Figure 3 – on Intermediate action triggered from line 68 – line 91

Text

Description automatically generated

When the user selects Intermediate level from the menu, the enum level is set to "2" and the relevant labels display to suit the status, but only the range in which the myRandomNumber() function is called changes because the game needs to give more sophisticated target values as the level hierarchy progresses.

Figure 4 – on Advanced action triggered from line 93 – line 120

Text

Description automatically generatedWhen the Advanced level is triggered, the range changes and the timer value is capped at 20 seconds, the same idea applies. Also, three more buttons are added to the level which were hidden during the previous levels.

Figure 5 – mytimer() function from line 122 – line 149

Explaining the main logic in this function, it updates the timer, which was set to 30 seconds, to timeout every second by reducing by one and displaying to the player, while a counter is used to keep track of how many times the timer was reduced by one, so when the counter reaches 30, the program sets the status to "Game Over," stops the timer, and disables all the buttons so that the player cannot keep clicking them. This status is displayed in red using a stylesheet to highlight that something is problematic at a glance. This happens only if the enum value of the level variable is either “1” or “2” meaning if the game is on either “Beginner” or “Advanced” levels.

In addition, because the timer in Advanced level is set to 20 seconds, the situation changes so that the counter monitors up to 20 counting and sends the same message to the user.

Text

Description automatically generated

Figure 6 – myRandomNumber() function from line 151 – line 238

The approach entails creating all of the player-interactive push button text values as well as the target value to be added up using them. The logic I used here is to generate an array of distinct values that are bounded by the given range start and end values, sum up the first three array element values, and display that value as the target value for the player to sum up, then shuffle all the element values in the array to reorder the arrangement, and finally set to text values of the push buttons for the player to choose from in order to make the target value.

Because only beginning and intermediate levels contain 6 option buttons, the array size, number of elements to be summed to form the goal value, and the number of button options displayed are provided correspondingly if the level enum represents either "1" or "2." If else, the player is presented with nine alternative buttons at the Advanced level, each of which modifies the facts indicated above.

Text

Description automatically generated

Figure 7 – on Again button Clicked from line 240 – line 290

When the player presses the play again button, it first determines which level the player is on; if it's level 1, it generates a new target number within the specified range, then resets the timer and reactivates all of the push buttons, as well as resetting the colors and starts the calculation process. The same principle applies to the other levels, which are recognized using if-else conditions based on the level enum value and adding level specific values.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 8 – on button clicks from line 293 onwards

The text value of each push button is extracted and allocated to variables by converting the value to an integer when the button is pressed. The sum of all the clicked push buttons is then calculated by adding those retrieved values to a total. Once a button has been pressed, it is disabled so that the player cannot use it again, and the method calculation() is then called to complete the logic. The same concept applies to each and every button names from btn0 – btn8.

Text

Description automatically generated

Figure 9 – calculation() function from line 383 – line 401

To explain the rest of the logic, this function accesses the total variable and checks whether that value is equal to the sum, which is the target value. If it is, the player is declared the winner, and all the buttons are disabled, and the timer is stopped. Otherwise, if the player clicks on button values that are more than the goal value, the game is lost, all buttons are disabled, and the timer is resumed.

Text

Description automatically generated

Figure 10 – button Disable/Enable, Resetting button color

The following source figures show the main function methods used to avoid code repetition, such as button disabling and enabling code lines and resetting button colors. For example, when the user has either won or lost the game, all of the buttons will be disabled, which is simple to implement by calling this function rather than hard coding it on every single slot and signal, and when the player presses play again, all of the buttons will be enabled by using this function. The same idea applies to changing the color of the buttons.

A screenshot of a computer

Description automatically generated with medium confidence Text

Description automatically generated

# Desired Improvements

For the time being, there are no known flaws or bugs with the Target Sum Application; nevertheless, I have highlighted the following as needed enhancements for future revisions:

* Notify player using an alert when the remaining time in the timer gets less than 10 seconds and onwards.
* When the player does not interact after 5 seconds, the program may provide a hint to the player on what move to make.
* Extend the application's tiers to include additional fascinating mechanisms, making the game more entertaining and fun.
* To make the user experience better and more creative, include animations and transitions between stages and at the end of game states.
* Making the software into a multiplayer game in which two players compete to sum up as quickly as possible in order to win.