COMPUTING WITH PLYMOUTH UNIVERSITY

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 that was encountered was "'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. Other than that, no issues were found when tested across platforms, albeit certain errors may occur owing to mismatches between versions of QT used, resulting in variances in library settings. Please note that this software program was produced and tested using QT creator 5.0.2 community.

User Documentation

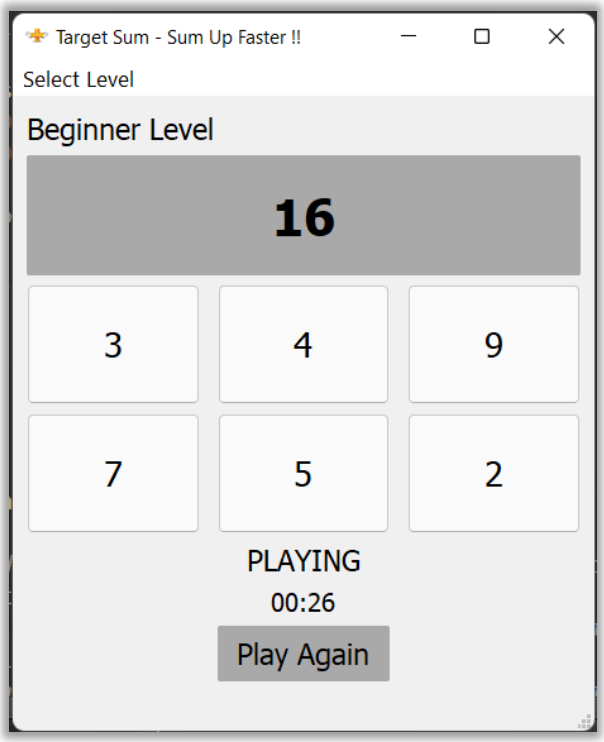
Description of the program

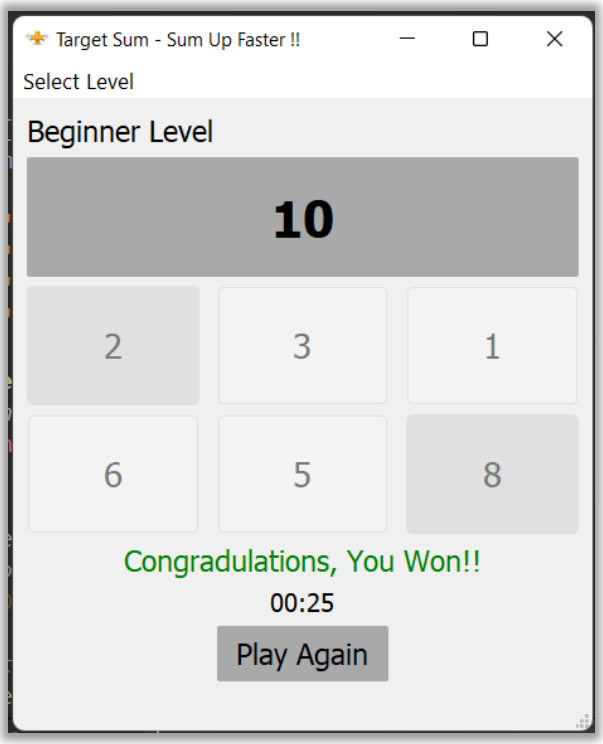
The software is a modest game that allows you to put your adding skills into action. It's time to sum up faster in order to win, as simple as that, once you've successfully extracted, built, and executed the application. The game is divided into three primary levels, which you can choose between what 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.

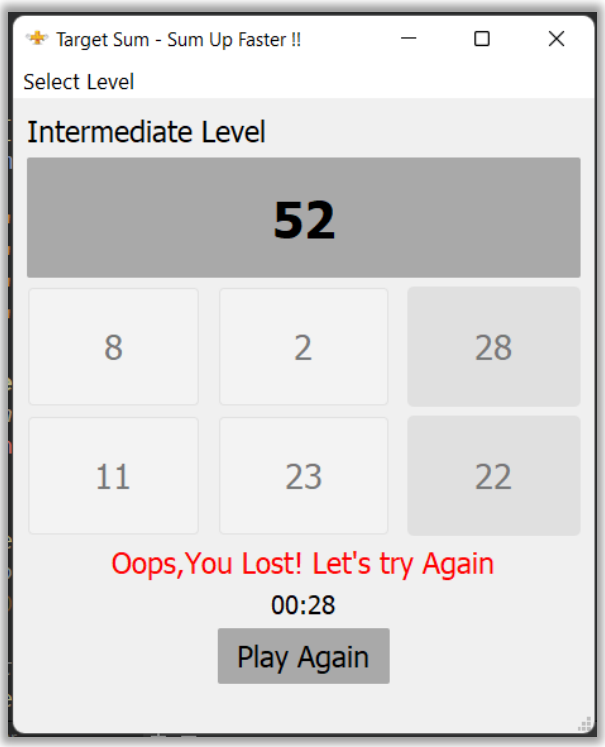
Supported Platforms

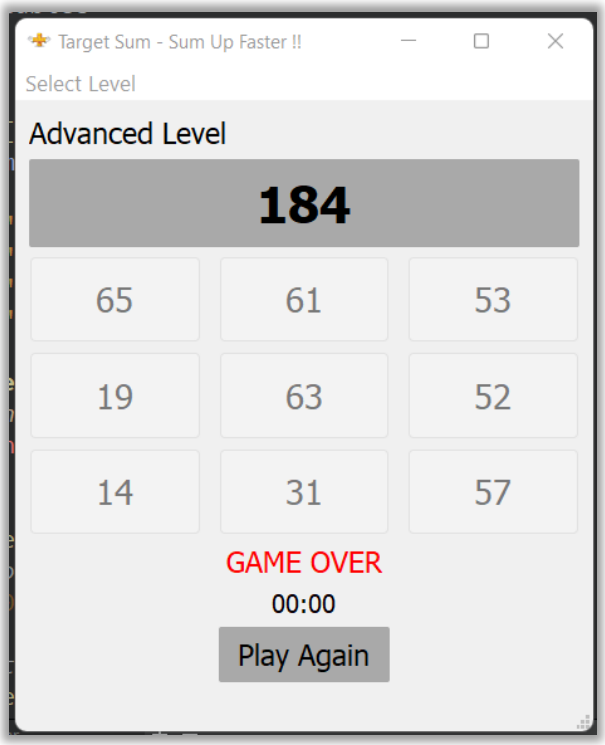
Target Sum QT Application has been tested and validated to work on Windows, Linux, and Mac OS at this time. On these platforms, screenshots of a game at various states and levels are shown below.

Windows 11 Version 21H2

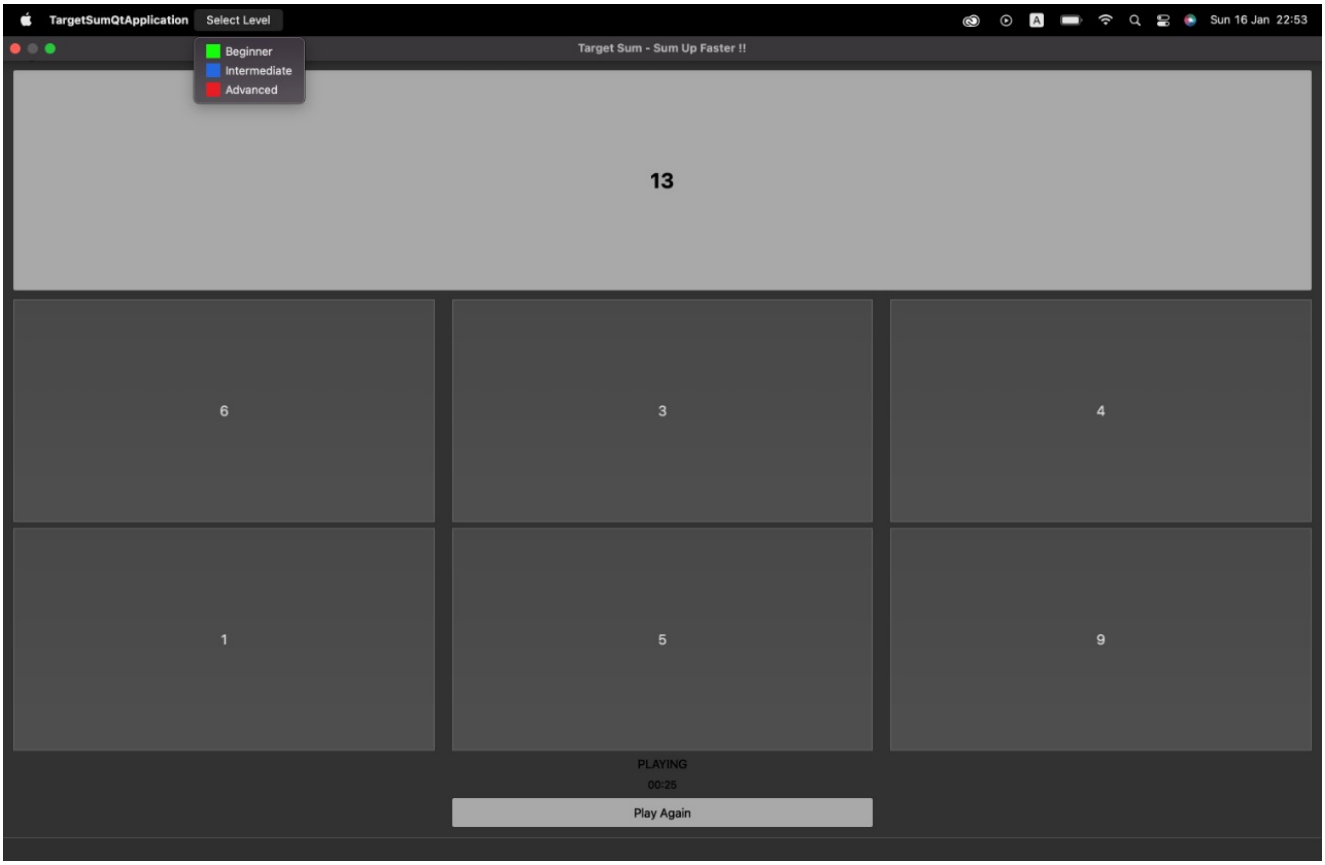








MacOS Monterey Version 12.1

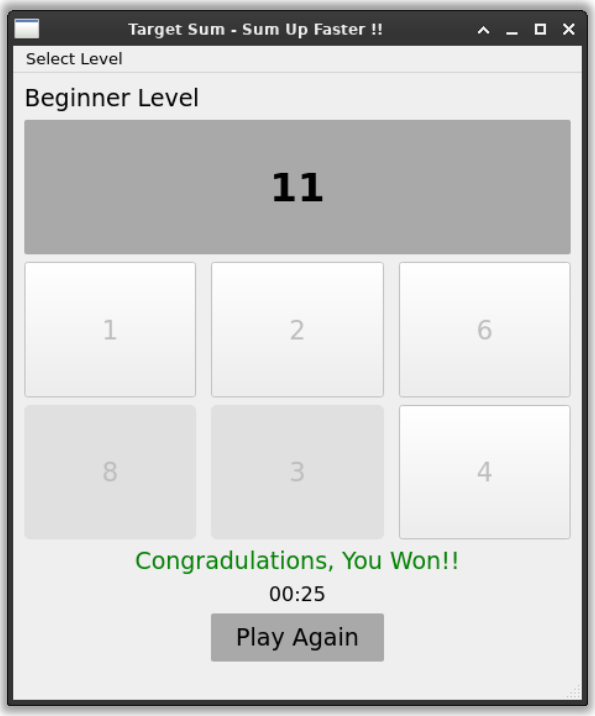


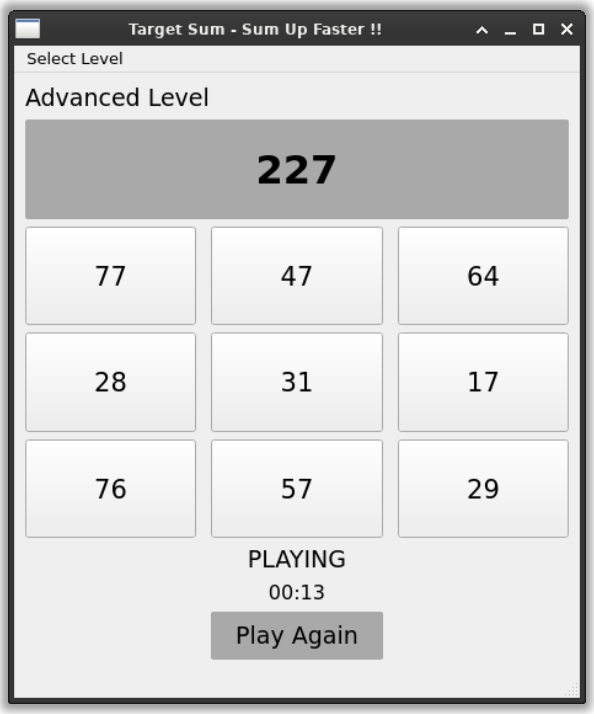
The Mac OS implementation resulted in serious User Interface inconsistencies, such as the clicked Styles of buttons being compromised with the default program backdrop, and the resizing properties not being applied in conjunction with the software package's menu bar. Despite the fact that the logic and 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.

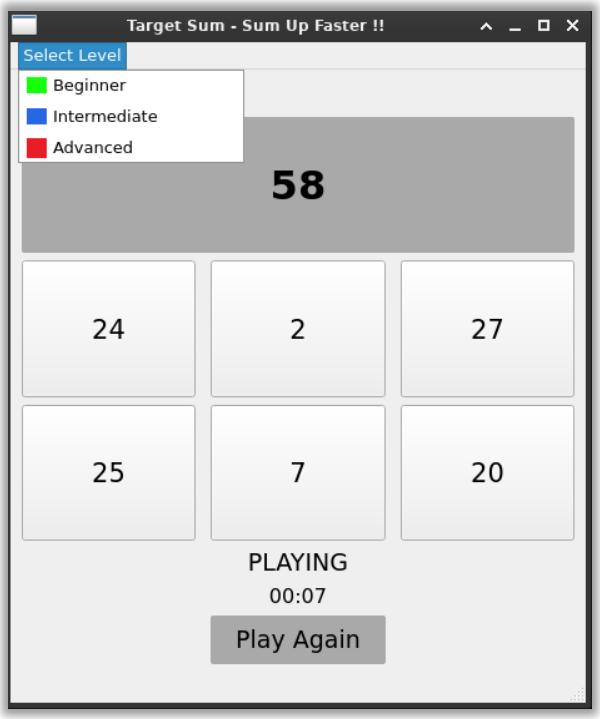
Screen Recording.mp4

* Ctrl + Click to Open the file.

Linux Debian verison 11

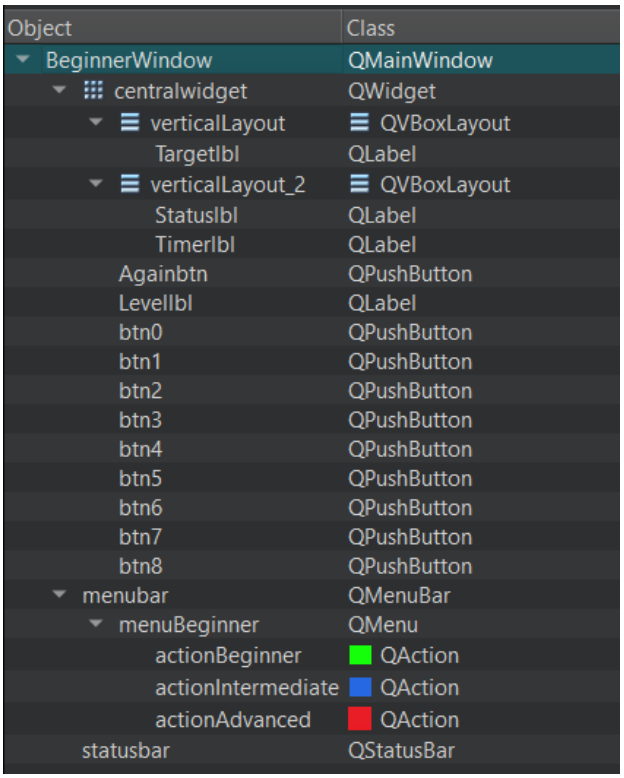




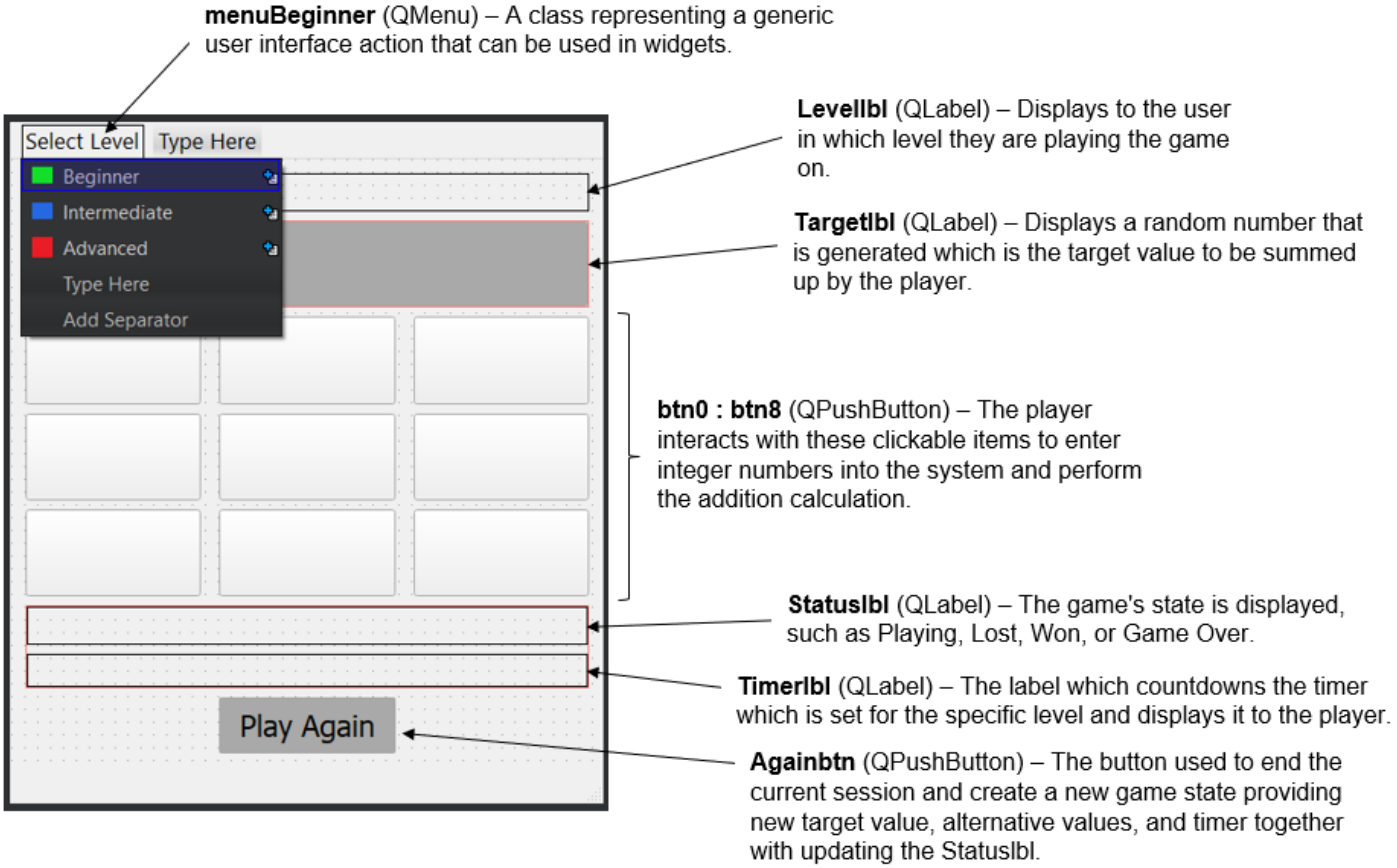


Technical Documentation

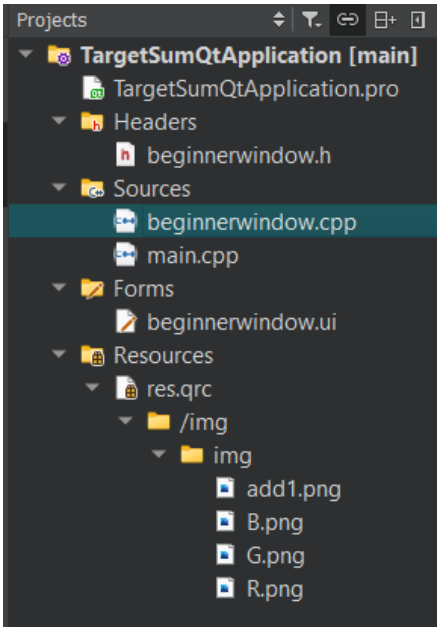
Description of the GUI design



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.
The 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



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.

```
17  BeginnerWindow::BeginnerWindow(QWidget *parent)
18      : QMainWindow(parent)
19      , ui(new Ui::BeginnerWindow)
20  {
21      ui->setupUi(this);
22      ui->Levellbl->setText("Beginner Level");
23      //Calling method to generate Random number and display it to the UI
24      myRandomNumber(1,10);
25
26      //Initialize "countdown" label text
27      ui->Timerlbl->setText("00:30");
28
29      //Connect timer to mytimer slot so it gets updated
30      timer = new QTimer();
31      connect(timer, SIGNAL(timeout()), this, SLOT(mytimer()));
32
33      //It is started with a value of 1000 milliseconds, indicating that it will time out every second.
34      timer->start(1000);
35      time.setHMS(0,0,30);
36  }
```

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.)

```
43  void BeginnerWindow::on_actionBeginner_triggered()
44  {
45      //enum used to detect the level
46      level=1;
47      //putting an end to the running timer
48      timer->stop();
49      ui->Levellbl->setText("Beginner Level");
50      //calls function to generate number between given range
51      myRandomNumber(1,10);
52      //setting counter to zero
53      counter = 0;
54      ui->Timerlbl->setText("00:30");
55      //starting the timer and setting it to 30 seconds
56      timer->start(1000);
57      time.setHMS(0,0,30);
58      //function used to update time together with the timer and counter
59      mytimer();
60      //Enables the pushbuttons
61      btnEnable();
62      //Calls the calculation function to start calculation of values
63      calculation();
64      //resets the color of the pushbuttons
65      ResetColor();
66  }
```

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

```
68  void BeginnerWindow::on_actionIntermediate_triggered()
69  {
70      //enum used to detect the level
71      level=2;
72      //putting an end to the running timer
73      timer->stop();
74      ui->Levellbl->setText("Intermediate Level");
75      //calls function to generate number between given range
76      myRandomNumber(1,30);
77      //setting counter to zero
78      counter = 0;
79      ui->Timerlbl->setText("00:30");
80      //starting the timer and setting it to 30 seconds
81      timer->start(1000);
82      time.setHMS(0,0,30);
83      //function used to update time together with the timer and counter
84      mytimer();
85      //Enables the pushbuttons
86      btnEnable();
87      //Calls the calculation function to start calculation of values
88      calculation();
89      //resets the color of the pushbuttons
90      ResetColor();
91  }
```

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

```
93  void BeginnerWindow::on_actionAdvanced_triggered()
94  {
95      //displays the hidden buttons
96      ui->btn6->show();
97      ui->btn7->show();
98      ui->btn8->show();
99      //enum used to detect the level
100     level=3;
101     //putting an end to the running timer
102     timer->stop();
103     ui->Levellbl->setText("Advanced Level");
104     //calls function to generate number between given range
105     myRandomNumber(10,80);
106     //setting counter to zero
107     counter = 0;
108     ui->Timerlbl->setText("00:20");
109     //starting the timer and setting it to 20 seconds
110     timer->start(1000);
111     time.setHMS(0,0,20);
112     //function used to update time together with the timer and counter
113     mytimer();
114     //Enables the pushbuttons
115     btnEnable();
116     //Calls the calculation function to start calculation of values
117     calculation();
118     //resets the color of the pushbuttons
119     ResetColor();
120 }
```

When 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.

```
122 void BeginnerWindow::mytimer(){
123     //updates time
124     time = time.addSecs(-1);
125     counter += 1;
126     ui->Timerlbl->setText(time.toString("mm:ss"));
127     ui->Statuslbl->setStyleSheet("color: black;");
128     ui->Statuslbl->setText("PLAYING");
129     //detects the current level and loops upto 30 seconds
130     if(level==1 || level==2){
131         if(counter == 30){
132             ui->Statuslbl->setStyleSheet("color: red;");
133             ui->Statuslbl->setText("GAME OVER");
134             timer->stop();
135             //Disables the pushbuttons
136             btnDisable();
137         }
138     }
139     else{
140         //detects the current level and loops upto 20 seconds
141         if(counter == 20){
142                 ui->Statuslbl->setStyleSheet("color: red;");
143                 ui->Statuslbl->setText("GAME OVER");
144                 timer->stop();
145                 //Disables the pushbuttons
146                 btnDisable();
147             }
148     }
149 }
```

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.

```
151 void BeginnerWindow::myRandomNumber(int rangeStart,int rangeEnd){
152     //detects the ongoing level and assignes values to pushbuttons using the created array.
153     if(level==1 || level ==2){
154     sum=0;
155     int newitem;
156     //Generates an array of distinct numbers
157     for(int i=0;i<6;i++)
158     {
159         bool unique;
160         do
161         {
162           unique=true;
163           newitem=QRandomGenerator::global()->bounded(rangeStart, rangeEnd);//Generate number between the given range
164           for(int i1=0;i1<i;i1++)
165           {
166              if(randArray[i1]==newitem)
167              {
168                unique=false;
169                 break;
170              }
171           }
172         }while(!unique);
173         randArray[i]=newitem;
174 
175     }
176     //Gets the sum of the first three array values
177     for(int i=0; i<3;i++){
178         sum+=randArray[i];
179     }
180     //Assign the sum to the Targetlbl
181     ui->Targetlbl->setText(QString::number(sum));
182     //Shuffles the order of elements in the array
183     std::random_shuffle(&randArray[0],&randArray[6]);
184     //Display array values in the UI pushbuttons
185     ui->btn0->setText(QString::number(randArray[0]));
186     ui->btn1->setText(QString::number(randArray[1]));
187     ui->btn2->setText(QString::number(randArray[2]));
188     ui->btn3->setText(QString::number(randArray[3]));
189     ui->btn4->setText(QString::number(randArray[4]));
190     ui->btn5->setText(QString::number(randArray[5]));
191     //only 6 buttons are shown in these levels so the rest are hidden
192     ui->btn6->hide();
193     ui->btn7->hide();
194     ui->btn8->hide();
195     }
196     else{
```

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.

```
240 void BeginnerWindow::on_Againbtn_clicked()
241 {
242     if(level==1){
243     total=0;
244     //Generates a new target number
245     myRandomNumber(1,10);
246     //Resets the timer
247     counter = 0;
248     ui->Timerlbl->setText("00:30");
249     timer->start(1000);
250     time.setHMS(0,0,30);
251     mytimer();
252     //Enables the pushbuttons
253     btnEnable();
254     //Calls the calculation function
255     calculation();
256     ResetColor();
257     }
```

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.

```
293 void BeginnerWindow::on_btn0_clicked()
294 {
295     //takes the value displayed in the pushbutton type cast to integer and assign to global variable
296     num1 = ui-> btn0->text().toInt();
297     //records the total of clicked button values to total variable.
298     total = total +num1;
299     //disables the pushbutton so it is not clickable once again
300     ui->btn0->setEnabled(false);
301     //Checks whether updated total variable is equal with the target value
302     calculation();
303     //gives a common style to all clicked pushbuttons
304     ui->btn0->setStyleSheet(clickedStyle);
305 }
```

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.

```
383 void BeginnerWindow::calculation(){
384     //Checks whether the clicked button values are equal to the Target value
385     if(total==sum){
386         ui->Statuslbl->setStyleSheet("color: green;");
387         ui->Statuslbl->setText("Congradulations, You Won!!");
388         //stops the timer
389         timer->stop();
390         //disables all buttons
391         btnDisable();
392     }
393     else if(total>sum){
394         ui->Statuslbl->setStyleSheet("color: red;");
395         ui->Statuslbl->setText("Oops,You Lost! Let's try Again");
396         //stops the timer
397         timer->stop();
398         //disables all buttons
399         btnDisable();
400     }
401 }
```

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.

```
359 void BeginnerWindow::btnDisable(){
360     //disables all buttons
361     ui->btn0->setEnabled(false);
362     ui->btn1->setEnabled(false);
363     ui->btn2->setEnabled(false);
364     ui->btn3->setEnabled(false);
365     ui->btn4->setEnabled(false);
366     ui->btn5->setEnabled(false);
367     ui->btn6->setEnabled(false);
368     ui->btn7->setEnabled(false);
369     ui->btn8->setEnabled(false);
370 }
371 void BeginnerWindow::btnEnable(){
372     //enables all buttons
373     ui->btn0->setEnabled(true);
374     ui->btn1->setEnabled(true);
375     ui->btn2->setEnabled(true);
376     ui->btn3->setEnabled(true);
377     ui->btn4->setEnabled(true);
378     ui->btn5->setEnabled(true);
379     ui->btn6->setEnabled(true);
380     ui->btn7->setEnabled(true);
381     ui->btn8->setEnabled(true);
382 }
```

```
432 void BeginnerWindow::ResetColor(){
433     //resets the color of all the buttons
434     ui->btn0->setStyleSheet("outline: none;");
435     ui->btn1->setStyleSheet("outline: none;");
436     ui->btn2->setStyleSheet("outline: none;");
437     ui->btn3->setStyleSheet("outline: none;");
438     ui->btn4->setStyleSheet("outline: none;");
439     ui->btn5->setStyleSheet("outline: none;");
440     ui->btn6->setStyleSheet("outline: none;");
441     ui->btn7->setStyleSheet("outline: none;");
442     ui->btn8->setStyleSheet("outline: none;");
443 }
```

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.