G. Einsendeaufgabe

Objektorientierte Programmierung mit C++ und Qt

Code:
CPBS 13E-XX1-N01

Name:	Vorname:
Rein	Wjatscheslaw
Postleitzahl und Ort:	**Straße:**
14193 Berlin	Cunostr. 54A
Studien- bzw. Vertrags-Nr.:	**Lehrgangs-Nr.:**
800463563	246

Fernlehrer/in:
Datum:
Note:
Unterschrift Fernlehrer/in:

Bitte reichen Sie Ihre Lösungen über die Online-Lernplattform ein oder schicken Sie uns diese per Post. Geben Sie bitte immer den Code zum Studienheft an (siehe oben rechts).

Inhaltsverzeichnis

1 Aufgabe:

Aufgabe:

Erweitern Sie die Meldung am Ende des Memory-Spiels um eine Ausgabe des Gewinners. Lassen Sie dazu noch einmal ausdrücklich in dem Dialog, der am Spielende erscheint, anzeigen, wer die meisten Paare gezogen hat.

10 Pkt.

Lösung:

Es wird davon ausgegangen, dass ein Tipp zur Lösung nur einen Vorschlag und nicht eine Forderung darstellt, wird in der Aufgabe eine eigene Variante implementiert. Die Funktion, die beide Symbolleisten und die Wiedergabeliste ein- bzw. ausblendet, ist in der Listing (**??**) zu sehen. Dabei wird in der Abhängigkeit von der Methode isChecked() eine Auswahl getroffen ob die entsprechende Elemente ein- bzw. ausgeschaltet, die Markierungen geändert werden und die Vollbilddarstellung aktiviert bzw. deaktiviert wird.

Um den sinnvollen Zusammenspiel von den betroffenen Elementen zu gewährleisten, sind in den zugehörigen Anzeige-Funktionen von der Symbolleisten und der Wiedergabeliste die Änderungen (siehe Listings **??**, **??** und **??**) vorzunehmen. Dafür sind die Methoden setChecked(0) um die Markierung von dem Vollbildmodus zu löschen und showNormal() für die Umschaltung in normale Bildmodus an den entsprechenden Stellen eingefügt. Die zusätzliche Umschaltung in normale Darstellung ist zwar nicht eindeutig in der Aufgabe gefordert, erscheint aber als eine sinnvolle Funktion.

Außerdem sind die Änderungen im Quellcode sowie in der Listings ausführlich durch Kommentare beschrieben.

```
1       class memoryspiel : public QWidget
2       {
3         ...
4
5         //Aufgebe 1: Deklaration von dem spielerText
6         QString spielerText;
7         //fuer das aktuell umgedrehte Paar
8         memorykarte *paar[2];
9         //fuer die Punkte
10        //Aufgabe 1: Deklaration von dem spielerPunkte
11        int menschPunkte, computerPunkte, spielerPunkte;
12
13        ...
14      };
15
```

Listing 1.1: Deklaration der Variablen bzw. Attributen im Headerdatei für die Aufgabe 1

```
1       /*****************************************************************
2       * void memoryspiel::karteOeffnen(memorykarte *karte)
3       *
4       * Funktion: Karten oeffnen
5       *****************************************************************/
6       void memoryspiel::karteOeffnen(memorykarte *karte)
7       {
8         ...
9
10        //haben wir zusammen 21 Paare?
11        //dann ist das Spiel vorbei
12        if (menschPunkte + computerPunkte == 21)
13        {
14          ...
15
16          //Aufgabe 1. Die Meldung beim Spielende wird um die Anzeige
17          //des Gewinners und von ihm gezogene Karten-Paaren ergaenzt.
18          if (menschPunkte > computerPunkte)
19          {
20            spielerText = "Mensch";
21            spielerPunkte = menschPunkte;
22          }
23          else
24          {
25            spielerText = "Computer";
26            spielerPunkte = computerPunkte;
27          }
28          //Die Anzeige wird um zusaetzliche Informationen wie die Bezeichnung des
29          //Spielers und entsprechende Anzahl der gezogenen Karten-Paare erweitert.
30          //bitte in einer Zeile eingeben
31          QString status = QString("%1 hat mit %2 Paaren gewonnen")
32          .arg(spielerText).arg(spielerPunkte);
33          QMessageBox::information(this, "Spielende", status);
34
35          ...
36        }
37      }
38
```

Listing 1.2: Erweiterung der Anzeige am Spielende

2 Aufgabe:

Aufgabe:

Erstellen Sie für das Memory-Spiel eine Anzeige des aktuellen Spielers. Lassen Sie dazu in einem Label unterhalb des Spielfelds anzeigen, ob gerade der Mensch zieht oder der Computer.

20 Pkt.

Lösung:

Eine mögliche Variante für die Lösung ist aus dem Listing **??** zu entnehmen.

```
1   class memoryspiel : public QWidget
2   {
3     ...
4
5     QLabel *labelTextMensch, *labelMensch, *labelTextComputer, *labelComputer;
6     //Aufgebe 2: Deklaration von labelSpieler und labelTextSpieler
7     QLabel *labelSpieler, *labelTextSpieler, *labelTextSpielstaerke;
8
9     ...
10  };
11
```

Listing 2.1: Deklaration der Variablen bzw. Attributen im Headerdatei für die Aufgabe 2

Um die Lautstärke über einen Symbol ein- bzw. auszuschalten wird aktuelle Value von dem Schieberegler und somit auch dem Drehfeld (da diese miteinander verknüpft sind) in der Variable iLautTemp (siehe Listing **??** Zeile 28) zwischengespeichert und bei dem erneuten Einschalten wird die Lautstärke auf gesicherte Wert zurückgesetzt (**??** Zeile 19). Dabei wird sichergestellt, dass Schieberegler und Drehfeld sich bei dem Ein- bzw. Ausschalten durch Symbol immer in der entsprechenden Stellung befinden.

Das zweite Teil der Aufgabe lässt sich einfach durch die Auswertung der einzustellenden Lautstärke (siehe Listing **??** Zeile 11) realisieren. Dabei dient die Variable neueLautstarke als Auswahlkriterium für die Anzeige des Ton-Symbols.

```
1   //der Konstruktor
2   memoryspiel::memoryspiel()
3   {
4     ...
5
6     //Aufgabe 2: Definition von einer Text-Label "Es zieht der"
7     labelTextSpieler = new QLabel(this);
8     labelTextSpieler->setText("Es zieht der");
9     labelTextSpieler->setGeometry(10, 500, 60, 25);
10
11    //Aufgabe 2: Definition von einer Text-Label und
```

```
12      //Setzen die Anzeige "Mensch" als Bezeichnung fuer den Spieler,
13      //der das Spiel beginnt.
14      labelSpieler = new QLabel(this);
15      labelSpieler->setText("Mensch");
16      labelSpieler->setGeometry(80, 500, 50, 25);
17
18      ...
19    }
20
```

Listing 2.2: Definition im Konstruktor für die Aufgabe 2

```
1   /*****************************************************************
2   * void memoryspiel::spielerWechseln()
3   *
4   * Funktion: Methode wechselt den Spieler
5   *****************************************************************/
6   void memoryspiel::spielerWechseln()
7   {
8     //wenn der Mensch an der Reihe war, kommt jetzt der Computer
9     if (spieler == 0)
10    {
11      spieler = 1;
12      //Aufgabe 2: Aenderung von dem Text-Label auf die dem aktuellen
13      //Spieler entsprechende Bezeichnung. In dem Fall "Computer"
14      labelSpieler->setText("Computer");
15      computerZug();
16    }
17    else
18    {
19      spieler = 0;
20      //Aufgabe 2: Aenderung von dem Text-Label auf die dem aktuellen
21      //Spieler entsprechende Bezeichnung. In dem Fall "Mensch"
22      labelSpieler->setText("Mensch");
23    }
24  }
25
```

Listing 2.3: Änderung von dem Text-Label für den aktuellen Spieler

3 Aufgabe:

Aufgabe:

Bauen Sie eine Schummelfunktion in das Memory-Spiel ein. Ergänzen Sie dazu eine Schaltfläche mit dem Text **Schummeln**. Beim Anklicken der Schaltfläche sollen alle noch nicht aufgedeckten Karten für eine bestimmte Zeit angezeigt und danach automatisch wieder umgedreht werden.

70 Pkt.

Lösung:

Für die Lösung wird wie vorgeschlagen eine Methode der Klasse QListWidget (siehe Listing **??** Zeilen 11, 18) angewendet, die eine Liste nach bestimmten Kriterien durchsucht und bei der Übereinstimmung die Liste von Treffern zurück liefert. Und wenn danach eine andere Methode Namens count() (Listing **??** Zeile 23) einen Zählerwert Null zurückgibt, kann man davon ausgehen, dass der Eintrag in der Liste nicht existiert und kann eingefügt werden.

```
class memoryspiel : public QWidget
{

  ...

  //Aufgabe 3: Deklaration der Schaltflaeche
  QPushButton *schummelnButton;

  //fuer den Timer
  //Aufgebe 3: Deklaration von dem timerSchummeln
  QTimer *timerUmdrehen, *timerSchummeln;

  ...

  private:
  ...
  //Aufgabe 3: Deklaration von Schummeln-Methode
  void schummelnKarten();

  //die Slots
  private slots:
  ...
  //Aufgabe 3: Deklaration von einem Slot fuer den Schummeln-Timer
  void timerSlotSchummeln();
  //Aufgabe 3: Deklaration von einem Slot fuer die Schummeln-Schaltflaeche
  void schummelnSlot();

  ...

};

```

Listing 3.1: Deklaration der Variablen bzw. Attributen sowie Slots und Methoden im Headerdatei für die Aufgabe 3

```
//der Konstruktor
memoryspiel::memoryspiel()
{
  ...

  //Aufgabe 3: Definition einer Schaltflaeche fuer eine Schummeln-Methode
  schummelnButton = new QPushButton(this);
  schummelnButton->setText("Schummeln");
  schummelnButton->setGeometry(300, 460, 70, 25);

  ...

  //Aufgabe 3: Definition von einem Timer fuer die Schummeln-Methode
  timerSchummeln = new QTimer(this);
  timerSchummeln->setSingleShot(true);

  ...

  //Aufgabe 3: Signal-Slots Verbindungen fuer eine Schummeln-Methode
  QObject::connect(timerSchummeln,SIGNAL(timeout()),this,SLOT(timerSlotSchummeln()));
  QObject::connect(schummelnButton, SIGNAL(clicked()),this,SLOT(schummelnSlot()));

  ...
}

```

Listing 3.2: Definition im Konstruktor für die Aufgabe 3

```
/*****************************************************************
* void memoryspiel::timerSlotSchummeln()
*
* Aufgabe 3
* Funktion: Slot fuer den Schummeln-Timer. Wenn die vorgegebene Zeit
* abgelaufen ist, sollen die Karten wieder umgedreht werden.
*****************************************************************/
void memoryspiel::timerSlotSchummeln()
{
  //Zeit ist abgelaufen, die Karten wieder umdrehen
  schummelnKarten();
}

/*****************************************************************
* void memoryspiel::schummelnSlot()
*
* Aufgabe 3
* Funktion: Slot fuer den Schaltflaeche-Schummeln. Wenn die Schaltflaeche
* "Schummeln" betaetigt wurde, sollen die Karten angezeigt (geoeffnet)
* und der Timer fuer spaetere Umdrehen gestartet werden.
*****************************************************************/
void memoryspiel::schummelnSlot()
{
  schummelnKarten();
  //Timer starten
  timerSchummeln->start(2000);
}

/*****************************************************************
* void memoryspiel::schummelnKarten()
*
* Aufgabe 3
* Funktion: Zwei verschachtelte FOR-Schleifen (siehe Heft 13E Seite 19)
* werden verwendet um die Karten umzudrehen. Die Pruefung bereits
* geoeffneten und im Spiel vorhandenen Karten uebernimmt die Methode
* karten[]->umdrehen() der Klasse memorykarte.
*****************************************************************/
void memoryspiel::schummelnKarten()
{
  //die Karten wieder umdrehen
  for (int zeile = 0; zeile < 7; zeile++)
    for (int spalte = 0; spalte < 6; spalte++)
      karten[(spalte * 7) + zeile]->umdrehen();
}
```

Listing 3.3: Zwei Slots und eine Methode für die Aufgabe 3

Listings