

Lösung zur Bewerbungsaufgabe

Anti-Tic-Tac-Toe (Qt / C++)

Projektbeschreibung

Dieses Projekt wurde im Rahmen einer Bewerbungsaufgabe für die Ausbildung zum ****Fachinformatiker für Anwendungsentwicklung**** bei ****adesso**** entwickelt.

Es handelt sich um eine Variante des klassischen Spiels Tic-Tac-Toe, bei der ein Spieler verliert, sobald er drei identische Symbole in einer Reihe, Spalte oder Diagonale platziert.

Spielregeln

- Zwei Spieler (X und O)
- Abwechselnde Spielzüge
- Standardmäßig ist eine Spielfeldgröße von 4×4 vorgegeben.
Zusätzlich wurde eine variable Spielfeldgröße von 3×3 bis 10×10 implementiert.
- Drei gleiche Symbole in einer Reihe, Spalte oder Diagonale führen zum Verlust
- Unentschieden, wenn das Spielfeld vollständig gefüllt ist

Funktionaler Umfang

- Grafische Benutzeroberfläche auf Basis von Qt (Widgets)
- Flexible Anpassung der Spielfeldgröße
- Anzeige der Spielregeln über eine Schaltfläche
- Neustart des Spiels mit gleicher oder neuer Spielfeldgröße
- Strukturierter, gut dokumentierter und wartbarer Quellcode

Eingesetzte Technologien

- Programmiersprache: C++
- Framework: Qt (Widgets)
- Entwicklungsumgebung: Qt Creator

Projektstruktur

— main.cpp	# Einstiegspunkt der Anwendung
— gamewidget.h	# Deklaration der Klasse GameWidget
— gamewidget.cpp	# Implementierung der Klasse GameWidget
— gamewidget.ui	# UI-Datei, erstellt mit Qt Designer
— README.md	# Projektbeschreibung und Nutzungshinweise

Verwendung des Projekts

Zur Ausführung des Projekts stehen zwei alternative Möglichkeiten zur Verfügung.

Alternative 1: Klonen des Git-Repositories

Das vollständige Projekt kann alternativ über folgendes GitHub-Repository bezogen werden:

https://github.com/lekoumboujoel/Anti_Tic-Tac-Toe

Alternative 2: Erstellung einer Qt-Widgets-Anwendung

1. Start von Qt Creator
2. Auswahl von „Datei → Neues Projekt“
3. Auswahl von „Anwendung → Qt Widgets Application“
4. Vergabe eines geeigneten Projektnamens (z. B. AntiTicTacToe)
5. Auswahl des Speicherorts
6. Auswahl des Build-Systems (CMake oder qmake)
7. Auswahl eines passenden Qt-Kits (z. B. Desktop Qt 6)
8. Auswahl der Basisklasse „QWidget“
9. Erstellung des Projekts

Anschließend können die entsprechenden Codeabschnitte aus diesem Dokument in die jeweiligen Projektdateien übernommen werden.

Ausführung des Programms

1. Öffnen des Projekts in Qt Creator
2. Auswahl eines geeigneten Qt-Kits
3. Kompilieren und Starten der Anwendung

4. Auswahl der gewünschten Spielfeldgröße

5. Beginn des Spiels

```
// -----main.cpp-----
#include <QApplication>
#include "gamewidget.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    GameWidget w;
    w.show();
    return app.exec();
}

//-----gamewidget.h-----

#pragma once
#include <QWidget>
#include <QPushButton>
#include <QVector>
#include <QChar>

class GameWidget : public QWidget
{
    Q_OBJECT

public:
    explicit GameWidget(int size = 4, QWidget *parent =
nullptr);

private slots:
    void handleButton();
    void showRules(); // Slot zum Anzeigen der Regeln

private:
    int SIZE;
```

```

    QVector<QVector<QPushButton*>> buttons;
    bool crossTurn;

    void setupUi();
    void resetGame();
    bool boardFull() const;
    bool hasThreeInARow(QChar symbol) const;
};

```

```

// -----gamewidget.cpp-----

```

```

#include "gamewidget.h"
#include <QGridLayout>
#include <QMessageBox>
#include <QInputDialog>
#include <QPushButton>
#include <QVBoxLayout>
#include <QDebug>

GameWidget::GameWidget(int size, QWidget *parent)
    : QWidget(parent), crossTurn(true), SIZE(size)
{
    setWindowTitle("Tic-Tac-Toe");

    QVBoxLayout *mainLayout = new QVBoxLayout(this);

    // Schaltfläche, um die Regeln jederzeit anzuzeigen
    QPushButton *rulesButton = new QPushButton("Spielregeln",
this);
    mainLayout->addWidget(rulesButton);
    connect(rulesButton, &QPushButton::clicked, this,
&GameWidget::showRules);

    // Spielfeld erstellen

    // Neue Spielfeldgröße auswählen
    bool ok;
    int newSize = QInputDialog::getInt(
        this,
        "Spielfeldgröße",
        "Geben Sie die Spielfeldgröße ein (3-10):",
        SIZE,
        3, 10, 1, &ok
    );
}

```

```

        );

    if (!ok) {
        close();
        return;
    }
    SIZE = newSize;
    setupUi();
}

void GameWidget::showRules()
{
    QString rules = "Willkommen bei Tic-Tac-Toe!\n\n"
                   "Regeln:\n"
                   "- Zwei Spieler: X und O\n"
                   "- Die Spieler spielen abwechselnd\n"
                   "- Drei gleiche Symbole in einer
Reihe/Spalte/Diagonale führen zum VERLUST\n"
                   "- Das Spiel endet unentschieden, wenn das
Spielfeld voll ist";

    QMessageBox::information(this, "Spielregeln", rules);
}

void GameWidget::setupUi()
{
    QGridLayout *gridLayout = new QGridLayout();
    buttons.resize(SIZE);

    for (int i = 0; i < SIZE; ++i) {
        buttons[i].resize(SIZE);
        for (int j = 0; j < SIZE; ++j) {
            QPushButton *btn = new QPushButton("", this);
            btn->setFixedSize(80, 80);
            btn->setFont(QFont("Arial", 24, QFont::Bold));
            btn->setStyleSheet(""); // keine Anfangsfarbe
            gridLayout->addWidget(btn, i, j);
            buttons[i][j] = btn;
            connect(btn, &QPushButton::clicked, this,
&GameWidget::handleButton);
        }
    }

    layout()->addItem(gridLayout);
}

```

```

void GameWidget::handleButton()
{
    QPushButton *btn = qobject_cast<QPushButton*>(sender());
    if (!btn || !btn->text().isEmpty())
        return;

    QChar symbol;
    if (crossTurn) {
        symbol = 'X';
        btn->setStyleSheet("background-color: #87CEEB;"); //
Himmelblau
    } else {
        symbol = 'O';
        btn->setStyleSheet("background-color: #90EE90;"); //
Hellgrün
    }
    btn->setText(symbol);

    if (hasThreeInARow(symbol)) {
        QMessageBox::information(this, "Spielende",
                                QString("Spieler %1 hat
verloren!\n(3 in einer Reihe)").arg(symbol));
        resetGame();
        return;
    }

    if (boardFull()) {
        QMessageBox::information(this, "Spielende",
"Unentschieden!");
        resetGame();
        return;
    }

    crossTurn = !crossTurn;
}

void GameWidget::resetGame()
{
    QMessageBox::StandardButton reply = QMessageBox::question(
        this,
        "Spiel beenden",
        "Möchten Sie mit derselben Spielfeldgröße neu starten?",
        QMessageBox::Yes | QMessageBox::No,
        QMessageBox::Yes
    );

    if (reply == QMessageBox::Yes) {

```

```

        // Neustart mit derselben Spielfeldgröße
        for (auto &row : buttons)
            for (auto &btn : row) {
                btn->setText("");
                btn->setStyleSheet("");
            }
        crossTurn = true;
    } else {
        // Neue Spielfeldgröße auswählen
        bool ok;
        int newSize = QInputDialog::getInt(
            this,
            "Spielfeldgröße",
            "Geben Sie die neue Spielfeldgröße ein (3-10):",
            SIZE,
            3, 10, 1, &ok
        );

        if (!ok) {
            close();
            return;
        }

        // Neues GameWidget mit der neuen Größe erstellen
        GameWidget *newGame = new GameWidget(newSize);
        newGame->show();

        // Altes Fenster schließen
        close();
    }
}

bool GameWidget::hasThreeInARow(QChar s) const
{
    // Zeilen überprüfen
    for (int i = 0; i < SIZE; ++i) {
        for (int j = 0; j <= SIZE - 3; ++j) {
            if (buttons[i][j]->text() == s &&
                buttons[i][j+1]->text() == s &&
                buttons[i][j+2]->text() == s)
                return true;
        }
    }

    // Spalten überprüfen
    for (int j = 0; j < SIZE; ++j) {
        for (int i = 0; i <= SIZE - 3; ++i) {

```

```

        if (buttons[i][j]->text() == s &&
            buttons[i+1][j]->text() == s &&
            buttons[i+2][j]->text() == s)
            return true;
    }
}

// Diagonalen überprüfen (links oben → rechts unten)
for (int i = 0; i <= SIZE - 3; ++i) {
    for (int j = 0; j <= SIZE - 3; ++j) {
        if (buttons[i][j]->text() == s &&
            buttons[i+1][j+1]->text() == s &&
            buttons[i+2][j+2]->text() == s)
            return true;
    }
}

// Diagonalen überprüfen (rechts oben → links unten)
for (int i = 0; i <= SIZE - 3; ++i) {
    for (int j = 2; j < SIZE; ++j) {
        if (buttons[i][j]->text() == s &&
            buttons[i+1][j-1]->text() == s &&
            buttons[i+2][j-2]->text() == s)
            return true;
    }
}

return false;
}

bool GameWidget::boardFull() const
{
    // Alle Reihen durchlaufen
    for (const auto &row : buttons) {
        // Jeden Button in der Reihe prüfen
        for (const auto &btn : row) {
            // Falls ein Feld leer ist, ist das Spielfeld nicht
voll
            if (btn->text().isEmpty())
                return false;
        }
    }
    // Kein leeres Feld gefunden → Spielfeld ist voll
    return true;
}

```