Fachhochschule Südwestfalen
University of Applied Sciences

schriftliche Ausarbeitung zum Modul Multimedia Programmierung
Prof. Dr Walther Roth
WS 19/20

Thema: Weekeewachee

Kostyantyn Baranov

E-Mail: baranov.kostyantyn@fh-swf.de

Martrikelnummer: 10060874

David Behrenbeck

E-Mail: behrenbeck.david@fh-swf.de

Martrikelnummer: 10039939

Mike Frank Peddinghaus

E-Mail: peddinghaus.mikefrank@fh-swf.de

Martrikelnummer: 10043548

Abgabe: 24.04.2020

Inhaltsverzeichnis

Technologiebeschreibung

Das Projekt wurde mithilfe von QT und dessen Komponenten wie quick für die grafische Darstellung und multimedia für die Audio-Dateien, sowie den OpenGL Bibliotheken für die Darstellung von 3D Elementen entwickelt.
Das Projekt wurde sowohl auf Windows, wie auch auf MacOS entwickelt und ist auf Windows, Linux und MacOS lauffähig.
Das Spiel hat sowohl eine deutsche, als auch eine englische Übersetzung und orientiert sich an der Spracheinstellung des Systems.

Anleitung

Startaufstellung

Jeder Spieler hat am Anfang die 4 Spielfiguren Stein, Schere, Papier und Brunnen auf seiner Grundreihe stehen. Eine feste Formation gibt es hierfür nicht, diese wird zufällig generiert.
Gezogen wird abwechselnd, dabei kann immer ein Feld in jede beliebige Richtung gegangen werden.Das heißt möglich sind vorwärts, schräg, seitwärts und rückwärts wie der König beim Schachspiel.

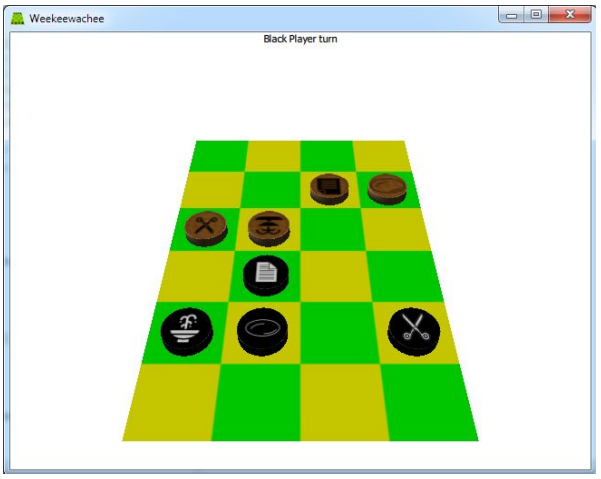
Die Startaufstellung:

Das Schlagen

Das Schlagen gilt nach den bekannten Regeln:

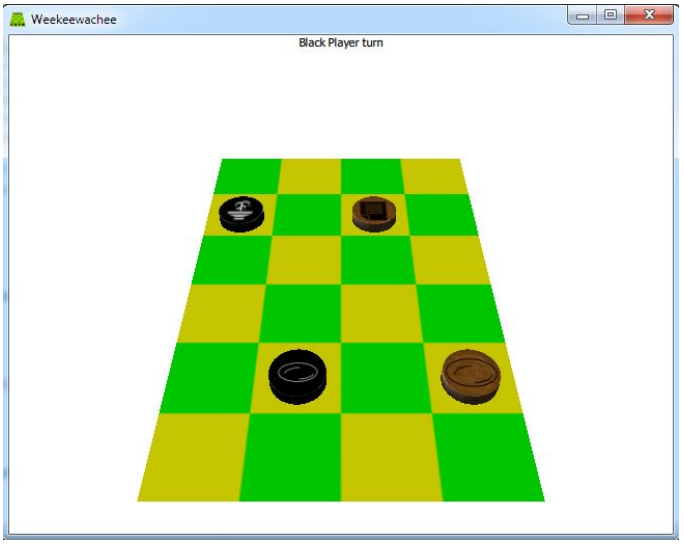
- Stein schlägt Schere
- Schere schlägt Papier
- Papier schlägt Stein und Brunnen
- Brunnen schlägt Stein und Schere.

In unserem Beispiel ist gerade der schwarze Spieler am Zug, welcher jetzt den weißen Brunnen schlagen kann.



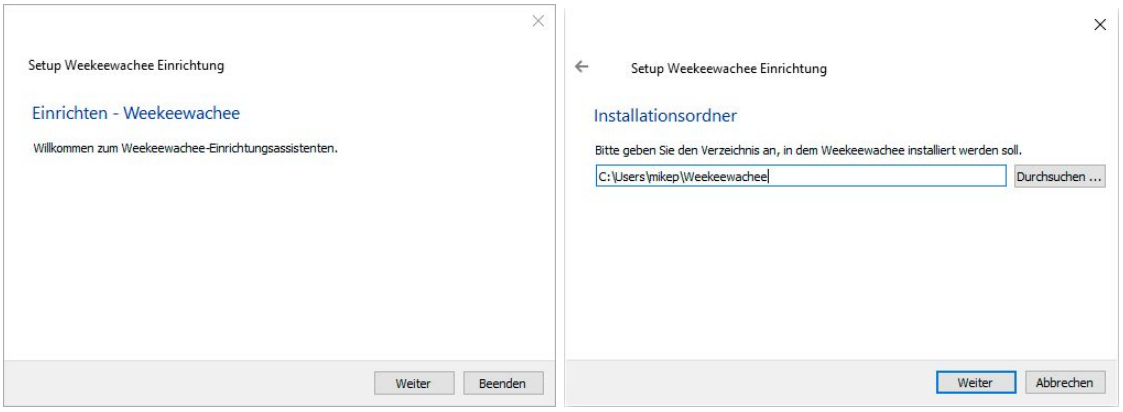
Spielende

Gewonnen hat, wer als Erstes eine seiner Figuren auf die gegnerische Grundreihe zieht oder alternativ alle gegnerischen Figuren schlägt.

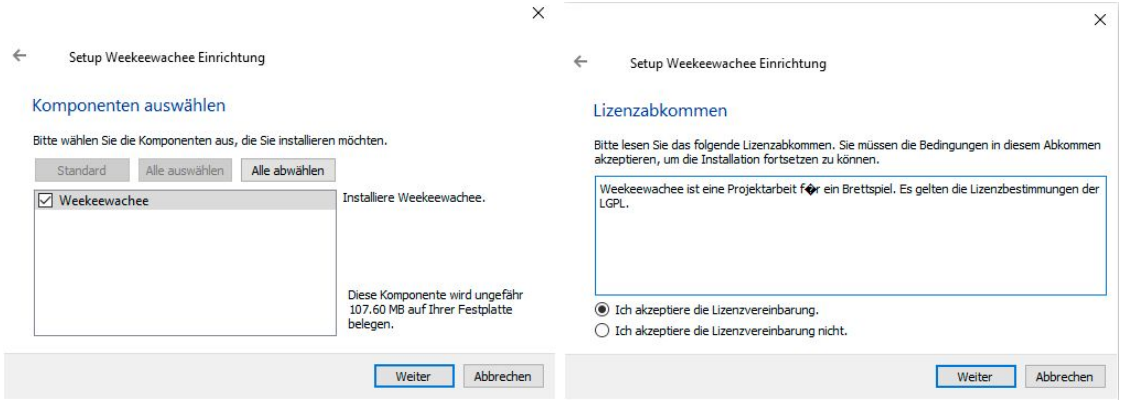


Der nächste Zug des schwarzen Spielers ist entscheidend, da dieser den Stein jetzt auf die Grundreihe des weißen Spieler stellen kann. Somit hat dieser gewonnen.

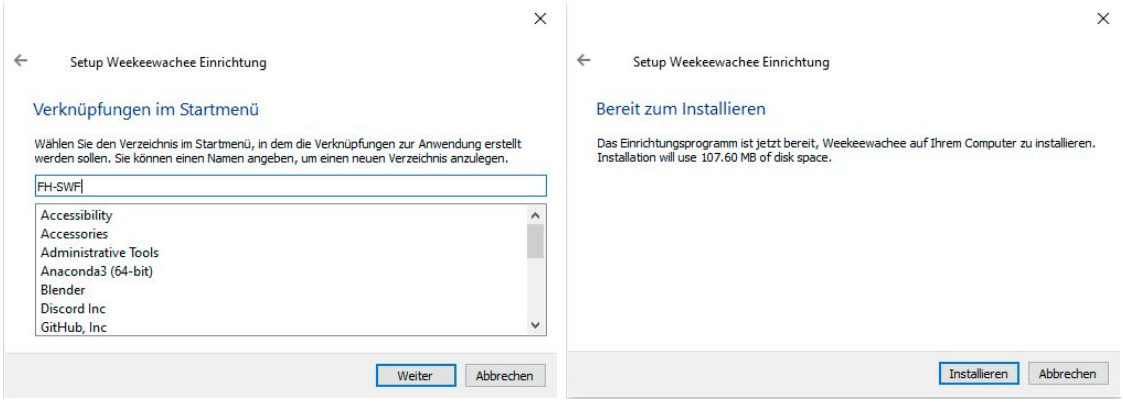
Installationsanleitung



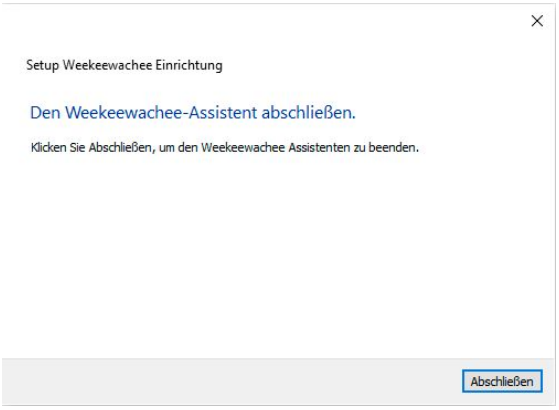
Für die Installation der Anwendung folgen Sie bitte dem Installations-Setup. Zunächst wählen Sie einen Speicherort aus, an welchem das Spiel installiert werden soll.



Als nächstes sollen Sie die Komponenten auswählen. Wählen Sie bitte die einzig vorhandene Komponente aus und akzeptieren Sie anschließend die Lizenzvereinbarung.



Zum Schluss können Sie noch ein Verzeichnis auswählen, in welches die Verknüpfung erstellt werden soll. Drücken Sie, nachdem Sie Ihr Verzeichnis gewählt haben, auf “Weiter” und anschließend auf “Installieren”.



Nachdem die Anwendung installiert wurde, drücken Sie auf “Abschließen”. Die Installation ist damit abgeschlossen.

Übersicht Anleitung

Spielstart

Zuerst wird in myglitem.cpp das Spiel erstellt. In myglitem.cpp werden sowohl das Spielfeld, als auch die Spielsteine initialisiert. Die Spielsteine sind GLDisc Objekte, das Spielfeld ein GLField Objekt.
Für jede Farbe werden 4 Spielsteine erstellt mit den Symbolen Stein, Schere, Papier und Brunnen. Jeder Stein hat dafür eine eigene Texture bekommen. Das Spielfeld besitzt ebenfalls eine eigene Textur.

Die Texturen werden auf diese Weise zugewiesen:

```
m_field->setTextureFile(":/textures/sbrett.png");
m_disc_white_schere->setTextureFile(":/textures/Stein_weiss_schere.png");
```

Nachdem alle Spielsteine und das Spielfeld seine Texturen erhalten haben, müssen nun die Model Files eingelesen werden.

```
m_disc_white_schere->readBinaryModelFile(":/models/Stein_weiss1.dat");
```

Diese Art Model wird von allen Steinen verwendet.

Danach werden alle Steine mit Hilfe von void MyGLItem::setDiscs() an einen zufälligen Platz bewegt:

```
void MyGLItem::setDiscs()
{
    QList<GLDisc*> blackdiscs = m_blackdiscs_list;
    QList<QVector3D> blackPositions = m_blackPos;
    QList<GLDisc*> whitediscs = m_whitediscs_list;
    QList<QVector3D> whitePositions = m_whitePos;
    // Shuffle
    qsrand(uint(time(nullptr)));
    for (int i = 0; i < 3; i++) {
        std::random_shuffle(whitePositions.begin()+i, whitePositions.begin()+4);
        std::random_shuffle(blackPositions.begin() + i, blackPositions.begin() + 4);
    }
    // Schwarze Steine
    for (int b = 0; b < blackdiscs.size(); b++) {
        blackdiscs[b]->move(blackPositions[b]);
        blackdiscs[b]->setHoldCoordinates(blackPositions[b]);
        blackdiscs[b]->setMoveCoordinates(blackPositions[b]);
        blackdiscs[b]->setXZ();
    }
    // Weiße Steine
    for (int w = 0; w < whitediscs.size(); w++) {
        whitediscs[w]->move(whitePositions[w]);
        whitediscs[w]->setHoldCoordinates(whitePositions[w]);
        whitediscs[w]->setMoveCoordinates(whitePositions[w]);
        whitediscs[w]->setXZ();
    }
}
```

Damit ist die Spielumgebung fertiggestellt und nun kann das Spiel begonnen werden.

Spielverlauf

Der Spielverlauf besteht darin, dass die Spieler zunächst Ihre Steine bewegen.

```
void MyGLItem::moving(GLDisc * disc, QVector3D MousePos)
{
    if (isMoveCorrect){
        alarmOff();
        QList<GLDisc*> friends_list;
        QList<GLDisc*> enemy_list;
        //Zuweisung von Listen
```

```
if (disc->getDisc_Color() == "black"){
    friends_list = m_blackdiscs_list;
    enemy_list = m_whitediscs_list;
} else {
    friends_list = m_whitediscs_list;
    enemy_list = m_blackdiscs_list;
}
float mouse_x = MousePos.x();
float mouse_z = MousePos.z();
QString buch = QString(disc->getDXZ_temp()[0]);
QString zahl = QString(disc->getDXZ_temp()[1]);
// Out of Range
if (mouse_x > 6.0f || mouse_x < -6.0f || mouse_z > 9.0f || mouse_z < -9.0f){
    showErrorMesage(tr("Out of Range!"));
    m_sounds->playSound(":/music/when.wav");
}
// X-Werte
if (mouse_x > -5.9f && mouse_x < -3.1f){
    buch = "A";
}
if (mouse_x > -2.9f && mouse_x < -0.1f){
    buch = "B";
}
if (mouse_x > 0.1f && mouse_x < 2.9f){
    buch = "C";
}
if (mouse_x > 3.1f && mouse_x < 5.9f){
    buch = "D";
}
// Z-Werte
if (mouse_z > 6.1f && mouse_z < 8.9f) {
    zahl = "1";
}
if (mouse_z > 3.1f && mouse_z < 5.9f) {
    zahl = "2";
}
if (mouse_z > 0.1f && mouse_z < 2.9f) {
    zahl = "3";
}
if (mouse_z > -2.9f && mouse_z < -0.1f) {
    zahl = "4";
}
if (mouse_z > -5.9f && mouse_z < -3.1f) {
    zahl = "5";
}
if (mouse_z > -8.9f && mouse_z < -6.1f) {
```

```
            zahl = "6";
        }
        //Überprüfung ob Maus bewegt wurde und ob das Feld, auf den die Maus zeigt, frei ist
        if ((disc->getDXZ_temp() != buch+zahl) && isFree(disc->getDXZ(), buch+zahl,
disc->getDisc_Name(), frends_list, enemy_list)){
            //Überprüfung ob Zielposition von Stein erreichbar ist
            if (disc->isMovementOk(buch+zahl)){
                //Wenn es nicht im gleichen Zug die erste Stein Bewegung ist, dann muss
backStep gemacht werden
                if(disc->isMoved()){
                    disc->backStep();
                }
                //Vektor um wie viel der Stein bewegt werden muss
                QVector3D movedisc = disc->getVector(disc->getList(), buch+zahl);
                //Zuweisung neuer Move Koordinaten
                disc->setMoveCoordinates(disc->getHoldCoordinates() + movedisc);
                disc->move(movedisc);
                //Aktualisierung temporärer Koordinaten
                disc->setDXZ_temp(buch+zahl);
                m_sounds->playSound(":/music/clearly.wav");
                //Stein ist in diesem Zug schon mindestens einmal beweget worden
                disc->setIsMoved(true);
            }
        }
    }
}
```

Dies geschieht abwechselnd.

Danach überprüfen wir, ob Platz frei ist.

```
bool MyGLItem::isFree(QString start, QString zelle, QString disc_name, QList<GLDisc*>
frends_list, QList<GLDisc*> enemy_list)
{
    QString stein = "";
    for (int i = 0; i < frends_list.size(); i++) {
        stein = frends_list[i]->getDXZ();
        if (stein == zelle && stein != start){
            showErrorMesage(tr("Same color!"));
            m_sounds->playSound(":/music/when.wav");
            return false;
        }
    }
    for (int i = 0; i < enemy_list.size(); i++) {
        stein = enemy_list[i]->getDXZ();
        if (stein == zelle && enemy_list[i]->getDisc_Name() == disc_name){
```

```
            showErrorMesage(tr("Same stone!"));
            m_sounds->playSound(":/music/when.wav");
            return false;
        }
    }
    return true;
}
```

Die fight Methode wird aufgerufen, wenn man die Maustaste loslässt.
Dabei sieht der Kampf wie folgt aus:

```
bool MyGLItem::figth(GLDisc *disc)
{
    if (disc->getDXZ() == disc->getDXZ_temp()){
        return false;
    }
    QList<GLDisc*> f_discs_list;
    QList<GLDisc*> e_discs_list;
    bool figth = false;
    // Listen
    if (disc->getDisc_Color() == "black"){
        f_discs_list = m_blackdiscs_list;
        e_discs_list = m_whitediscs_list;
    } else {
        f_discs_list = m_whitediscs_list;
        e_discs_list = m_blackdiscs_list;
    }
    // Kampf
    qDebug() << "Kampf";
    for (int i = 0; i < e_discs_list.size(); i++) {
        if(e_discs_list[i]->getHoldCoordinates() == disc->getMoveCoordinates() &&
disc->isFigth(e_discs_list[i]->getDXZ()))
        {
            if(disc->getDisc_Name() == "stein" && e_discs_list[i]->getDisc_Name() ==
"schere"){
                qDebug() << "Stein gegen Schere";
                move_away(e_discs_list[i]);
                e_discs_list = deletediscFromList(e_discs_list, e_discs_list[i]->getDisc_Name());
                figth = true;
                break;
            }
            //Gekürzt aus Gründen der Übersichtlichkeit
        }
    }
    // Liste aktualisieren
    if (figth) {
```

```
        if (disc->getDisc_Color() == "black"){
            m_whitediscs_list = e_discs_list;
            m_blackdiscs_list = f_discs_list;
        } else {
            m_blackdiscs_list = e_discs_list;
            m_whitediscs_list = f_discs_list;
        }
    }
    // set Moved
    disc->setIsMoved(false);
    // Koordinaten setzen
    disc->setHoldCoordinates(m_disc->getMoveCoordinates());
    // Update XZ
    disc->setStepVector(QVector3D(0.0f, 0.0f, 0.0f));
    disc->updateXZ();
    turnEnd();
    return true;
}
```

Nach einem Kampf wird der Sieger des Kampfes ermittelt. Durch die Funktionen void MyGLItem::move_away(GLDisc *disc) und QList<GLDisc *> MyGLItem::deleteDiscFromList(QList<GLDisc *> m_discs_list, QString disc_name) werden die Spielsteine aus dem Spiel genommen.

Mit der Funktion MyGLItem::move_away(GLDisc *disc) wird der Spielstein an die Stelle QVector3D(+100.0f, 0.0f, +100.0f) geschoben und ist somit außerhalb des Spielbereiches.

```
void MyGLItem::move_away(GLDisc *disc)
{
    disc->move(QVector3D(+100.0f, 0.0f, +100.0f));
    disc->setHoldCoordinates(disc->getHoldCoordinates() + QVector3D(+100.0f, 0.0f,
+100.0f));
    m_sounds->playSound(":/music/Blop.wav");
    qDebug() << "Disc " << disc->getDisc_Color() << " " << disc->getDisc_Name() << " ist
gelöscht";
}
```

Die Funktion QList<GLDisc *> MyGLItem::deleteDiscFromList(QList<GLDisc *> m_discs_list, QString disc_name) wird benötigt, um den Spielstein aus der Liste mit den aktuell verwendbaren Spielsteinen zu entfernen.

```
QList<GLDisc *> MyGLItem::deletediscFromList(QList<GLDisc *> m_discs_list, QString
disc_name)
{
    for (int i = 0; i < m_discs_list.size(); i++) {
        if (m_discs_list[i]->getDisc_Name() == disc_name){
```

```
            m_discs_list.removeAt(i);
        }
    }
    return m_discs_list;
}
```

Nachdem der Kampf zu Ende ist, ist auch die Spielrunde vorbei. Sollte es zu keinem Kampf gekommen sein, wird der Stein auf den freien Platz gesetzt.

Runde zu Ende

Nach jedem Spielzug wird die Funktion void MyGLItem::turnEnd() ausgeführt. In dieser wird überprüft, ob das Spiel vorbei ist oder nicht. Ist dies nicht der Fall, so wird die Farbe gewechselt und das Spielbrett um 180 Grad gedreht.

```
void MyGLItem::turnEnd()
{
    if (isGameOver()){
        restartGame();
    }  else {
        // Actual disc to Fake disc
        m_disc = m_disc_other;
        // Spieler wechsel
        changePlayer(player);
        // Board umdrehen
        rotateBoard();
    }
}
```

Spielende

Nach jedem Spielzug wird überprüft, ob das Spiel zu Ende ist und ob es einen Gewinner gibt. Dafür wird überprüft, ob entweder einer der Steine in der Endzone des anderen Spielers liegt, oder ob eine der beiden Listen leer ist.

```
bool MyGLItem::isGameOver()
{
    //Überprüfung ob ein schwarzer Stein in den Startpositionen der weißen Steine ist
    for (int b = 0; b < m_blackdiscs_list.size(); b++) {
        for (int i = 0; i < 4; i++) {
            if (m_blackdiscs_list[b]->getMoveCoordinates() == m_whitePos[i]){
                qDebug() << "Schwarzer Spieler gewinnt!";
                m_sounds->playSound(":/music/applauses.wav");
                return true;
            }
```

```
        }
    }
    //Überprüfung ob ein weißer Stein in den Startpositionen der schwarzen Steine ist
    for (int w = 0; w < m_whitediscs_list.size(); w++) {
        for (int j = 0; j < 4; j++) {
            if (m_whitediscs_list[w]->getMoveCoordinates() == m_blackPos[j]){
                qDebug() << "Weißer Spieler gewinnt!";
                m_sounds->playSound(":/music/applauses.wav");
                return true;
            }
        }
    }
    //Überprüfung ob es noch weiße Spielsteine gibt
    if (m_whitediscs_list.isEmpty()){
        qDebug() << "Schwarzer Spieler gewinnt!";
        m_sounds->playSound(":/music/applauses.wav");
        return true;
    }
    //Überprüfung ob es noch schwarze Spielsteine gibt
    if (m_blackdiscs_list.isEmpty()){
        qDebug() << "Weißer Spieler gewinnt!";
        m_sounds->playSound(":/music/applauses.wav");
        return true;
    }
    return false;
}
```

Übersichtsdokumentation

Für das Projekt haben wir uns am Praktikum orientiert und die dort vorhandenen Klassen abgeändert. Die Klasse mit den meisten Änderungen ist die Klasse myGLItem, die nahezu komplett überarbeitet wurde. Die wichtigsten Methoden wurden in der Ausarbeitung bereits näher erläutert. Zusätzlich gibt es die neue Klasse music, in der wir die Soundeffekte implementiert haben. Außerdem haben wir die Klasse GLDisc um einige Funktionen erweitert, damit Sie unseren Ansprüchen entspricht und die benötigten Funktionen für uns erfüllt.

Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Ausarbeitung selbständig erarbeitet und alle verwendeten Hilfsmittel angegeben haben.

28.04.2020
Datum

Kostyantyn Baranov

Behrenbeck

David Behrenbeck

Mike Peddinghaus

Mike Frank Peddinghaus

Literaturverzeichnis

Praktikumsunterlagen Multimedia-Programmierung 2019/20
QT Docs: https://doc.qt.io/