

Entwurfsarbeit

„Qwixx“

Grafische Nutzerschnittstellen, WiSe 2019/20

Unter der Leitung von Dipl.-Inf., Dipl.-Ing (FH) Michael Wilhelm

Autor:

Oliver Lindemann
Hochschule Harz
University of Applied Science
Matrikelnummer: 26264
u33873@hs-harz.de

Abgabedatum: 01.03.2020

Inhalt

1	Problembeschreibung	3
1.1	Projektziel	3
1.2	Problemstellung	3
1.3	Motivation	4
	Implementierung	4
1.4	Aufbau	4
1.5	Beschreibung der Klassen	5
1.5.1	Die Würfel (Dice)	5
1.5.2	Der Spieler (Player)	6
1.5.3	Das Spielfeld (GameBoard)	7
1.5.4	Das Spiel (Game)	8
2	Signifikante Implementierungsdetails	9
2.1	Abschließen von Reihen	9
2.2	GameThread	9
2.2.1	Auswahl eines Würfels	10
2.3	Auf Eingabe von Benutzer warten	11
2.4	Darstellen eines Würfels	12
3	Das User Interface (UI)	13
3.1	Aufbau des Spielfeldes	13
3.2	Der gesamte Spielblock im Detail	15
3.3	Darstellung der Würfel	16
3.4	Anordnung mehrerer Spielfelder	17
4	JUnit-Tests	18
5	Verwendete Bibliotheken	18
6	Erweiterung	19
6.1	Funktionale Erweiterung	19
6.2	Grafische Erweiterung	19
7	Quellenverzeichnis	20

1 Problembeschreibung

1.1 Projektziel

Das Ziel dieses Projektes ist es, das taktische Würfelspiel „Qwixx“ als lauffähiges Computerspiel zu realisieren.

1.2 Problemstellung

Es soll das Gesellschaftsspiel „Qwixx“ als digitale Version entwickelt werden. Dieses Spiel muss mit mindestens zwei, aber maximal mit fünf Personen gespielt werden. Die Mitspieler werden in der digitalen Fassung durch Computergegner simuliert.

Es gibt kein gemeinsam genutztes Spielbrett, sondern jeder Mitspieler erhält zu Beginn eines jeden Spiels sein eigenes Spielfeld bzw. seinen eigenen Spielblock. Dieser besteht aus vier unterschiedlich eingefärbten Reihen mit jeweils elf Zahlenfeldern von zwei bis zwölf (oder in umgekehrter Reihenfolge) und einem Bonusfeld. Zusätzlich zu den Spielblöcken gibt es sechs Würfel, welche aus zwei weißen und jeweils einem roten, gelben, grünen und blauen Würfel bestehen.

Ziel des Spiels ist es, auf seinem eigenen Spielblock möglichst viele Felder in den vier (Farb-) Reihen anzukreuzen. Je mehr Kreuze ein Spieler in einer Farbreihe hat, desto mehr Punkte erhält er dafür. Gewonnen hat derjenige, der am Ende insgesamt die meisten Punkte hat.

Auf dem Spielblock dürfen die Felder nur von links nach rechts angekreuzt werden. Dabei muss allerdings nicht zwingend beim ersten Feld begonnen werden. Zudem dürfen Felder auch übersprungen werden. Diese ausgelassenen Zahlenfelder dürfen nachträglich allerdings nicht mehr angekreuzt werden.

Gespielt wird im Uhrzeigersinn. Der aktive Spieler würfelt mit allen sechs Würfeln. Nun werden zwei Aktionen hintereinander ausgeführt.

Aktion 1: Die Augen der beiden weißen Würfel werden addiert. Die Summe darf nun jeder Spieler (muss aber nicht) in einer beliebigen Farbreihe auf seinem Spielblock ankreuzen.

Aktion 2: Der aktive Spieler darf jetzt als einziger noch die Farbwürfel verwenden. Hierbei kombiniert er einen Farbwürfel seiner Wahl mit einem der beiden weißen Würfel (ebenfalls frei wählbar). Von beiden ausgesuchten Würfeln addiert er nun die Augenzahl und trägt die Summe in die Farbreihe des gewählten Farbwürfels ein.

Falls der aktive Spieler keine der beiden Aktionen ausführen möchte oder kann, so muss dieser auf seinem Spielblock in der Spalte „Fehlwurf“ ein Kreuz setzen.

Nicht-aktive Spieler, die lediglich die erste Aktion ausführen durften, müssen allerdings kein Kreuz bei den Fehlwürfen machen, falls diese die erste Aktion nicht ausführen wollten oder konnten.

Nach Abschluss der beiden Aktionen ist der nächste Spieler an der Reihe. Dieser würfelt nun erneut mit allen sechs Würfeln. Danach werden wieder beide oben beschriebenen Aktionen ausgeführt.

Das Spiel endet, sobald ein Mitspieler alle vier Fehlwürfe angekreuzt hat oder eine zweite Farbreihe abgeschlossen wurde.

Eine Farbreihe kann abgeschlossen werden, indem die letzte Zahl in einer Reihe angekreuzt wird und insgesamt mindestens sechs Kreuze in dieser Reihe gesetzt wurden. Sobald also ein Mitspieler das letzte Feld einer Reihe ankreuzt und dieses Kreuz das mindestens sechste in dieser Farbreihe ist, so beendet er diese Reihe und darf das Zusatzfeld neben der Reihe (durch ein Schloss gekennzeichnet) ebenfalls ankreuzen.

Die abgeschlossene Reihe ist nun auch für alle anderen Spieler geschlossen und es dürfen keine weiteren Kreuze in dieser Reihe gemacht werden.

Nach Beenden der zweiten Farbreihe oder dem Ankreuzen aller Fehlwürfe errechnen alle Mitspieler ihre Punktzahl. Der Spieler mit der höchsten Punktzahl gewinnt.

1.3 Motivation

Das Gesellschaftsspiel „Qwixx“ existiert nicht in digitaler Form, wobei es hierfür sehr gut geeignet wäre. Eine digitale Fassung bietet die Möglichkeit, dieses Spiel allein gegen Computergegner zu spielen und erfordert keine anderen menschlichen Mitspieler. Zudem werden keine Spielmaterialien wie z.B. Würfel oder Spielblöcke benötigt. Die einzige Voraussetzung ist ein Computer mit einer passenden installierten JAVA-Version.

Implementierung:

1.4 Aufbau

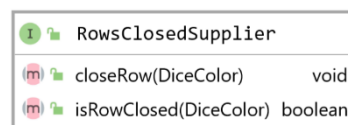
Ein Spiel (*Game*) besteht aus:

- 6 Würfeln (*IDice*) und die daraus kombinierten Würfelpaare (*DicePair*)
- Mehreren Mitspielern (*IPlayer*)

Ein Spieler hat genau ein Spielfeld (*GameBoard*), welches wiederum aus einem Score (*UserScore*) und verschiedenfarbigen Reihen (*Row*) besteht. Jede Reihe hat zudem mehrere Felder (*Fields*).

Das Spiel (*Game*) implementiert zudem noch das Interface *RowClosedSupplier*, welches Methoden zur Abfrage von abgeschlossenen und zum Schließen von Reihen zur Verfügung stellt.

Dieser *RowClosedSupplier* wird von jedem Spielfeld (*GameBoard*) benötigt, um eine Reihe abzuschließen.



Die **Abbildung 1 Aufbau Qwixx Model - UML** gibt einen Überblick über die Relation der einzelnen Klassen zueinander. Diese werden im Folgenden detailliert beschrieben.

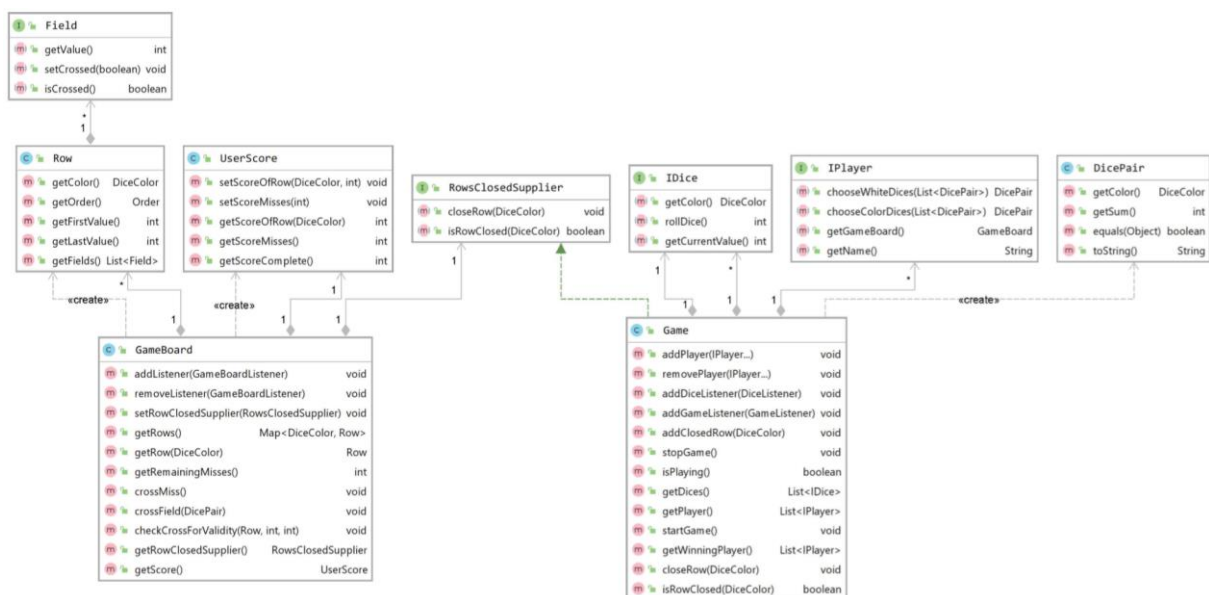
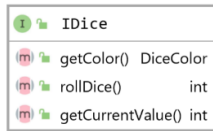


Abbildung 1 Aufbau Qwixx Model - UML

1.5 Beschreibung der Klassen

1.5.1 Die Würfel (Dice)

Die Würfel in diesem Spiel repräsentieren nicht nur eine zufällige Augenzahl, sondern besitzen auch eine Farbe (DiceColor).



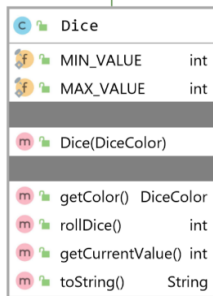
1.5.1.1 IDice

Das Interface IDice stellt drei Methoden zur Verfügung.

getColor() – liefert die Farbe dieses Würfels

rollDice() – würfelt diesen Würfel und liefert den geworfenen Wert

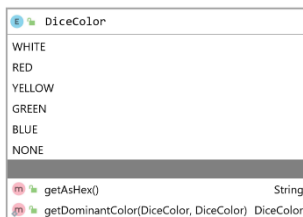
getCurrentValue() – Liefert den mit diesem Würfel zuletzt gewürfelten Wert



1.5.1.2 Dice

Diese Klasse implementiert das Interface IDice.

Über die Konstanten *MIN_VALUE* und *MAX_VALUE* stellt diese Klasse zudem den minimalen- und maximalen Wert zur Verfügung, der mit diesem Würfel geworfen werden kann.

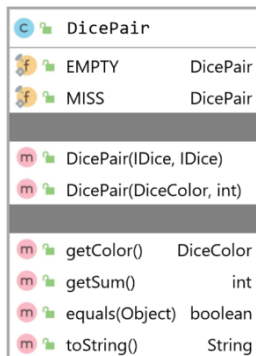


1.5.1.3 DiceColor

Das Enum DiceColor beinhaltet verschiedene Farben, die ein Würfel haben kann. Dieses Enum beinhaltet noch zwei Methoden:

getAsHex() – liefert die Würfelfarbe in Hex-Darstellung

getDominantColor() – liefert die dominante Farbe der gegebenen Farben



1.5.1.4 DicePair

Die Klasse DicePair stellt zwei Würfel (IDice) als Paar dar. Ein Würfelpaar hat eine bestimmte Farbe (die dominante Farbe zweier Würfel) und eine Würfelsumme (Augenzahl zweier Würfel addiert).

EMPTY – repräsentiert leeres Würfelpaar

MISS – repräsentiert einen Fehlwurf

getColor() – liefert die Farbe dieses Würfelpaares

getSum() – liefert die Summe dieses Würfelpaares

1.5.2 Der Spieler (Player)

1.5.2.1 IPlayer

Ein Spieler hat einen Namen und ein Spielfeld. Zudem kann er über die Methode *chooseDicePair* (*List<DicePair>*, ...) gefragt werden, welches Würfelpaar er aus der gegebenen Liste auswählen und auf seinem Spielfeld verwenden möchte.



getName() – liefert den Namen des Spielers

getGameBoard() – liefert das Spielfeld des Spielers

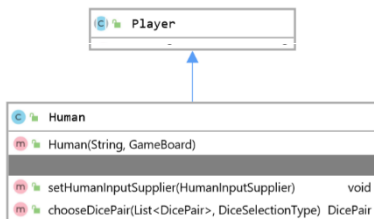
chooseDicePair(...) – fordert den Spieler auf, ein Würfelpaar aus der gegebenen Liste auszuwählen.



1.5.2.2 Player

Die abstrakte Klasse Player implementiert die Methoden *getName()* und *getGameBoard()* des Interfaces IPlayer.

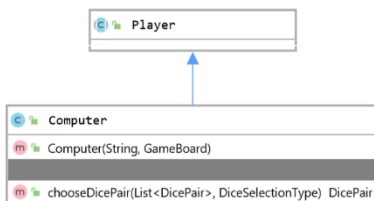
Die Methode *chooseDicePair()* muss jedes Kind dieser Klasse noch selbst implementieren.



1.5.2.2.1 Human

Die Klasse Human repräsentiert den menschlichen Spieler und erbt von der Klasse Player.

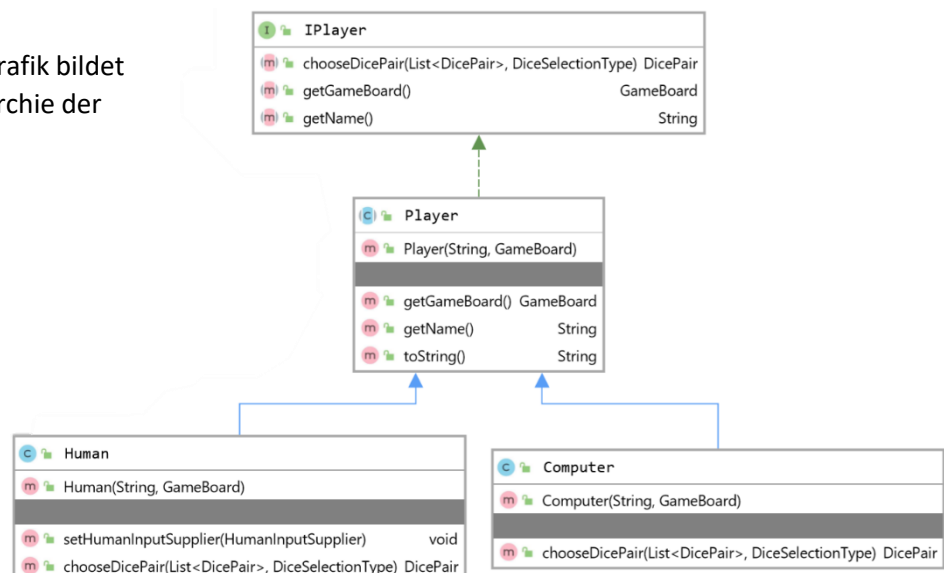
setHumanInputSupplier(...) – setzt für diesen menschlichen Spieler den *HumanInputSupplier*. Dieser entscheidet über die Auswahl des zu wählenden Würfelpaares.



1.5.2.2.2 Computer

Die Klasse Computer repräsentiert einen Computerspieler und erbt von der Klasse Player. Diese Klasse stellt keine weiteren Methoden zur Verfügung.

Die nebenstehende Grafik bildet die vollständige Hierarchie der Player-Klassen ab.



1.5.3 Das Spielfeld (GameBoard)

1.5.3.1 GameBoard

Das Spielfeld beinhaltet eine Liste von Reihen, den UserScore und die Anzahl der verbleibenden Fehlwürfe. Ein GameBoard benötigt zudem noch einen RowClosedSupplier, der über den entsprechenden Setter gesetzt werden kann.

GameBoard	
MIN_FIELDS_CROSSED_TO_CLOSE_ROW	int
AMOUNT_MISSES	int
GameBoard()	
addListener(GameBoardListener)	void
removeListener(GameBoardListener)	void
setRowClosedSupplier(RowsClosedSupplier)	void
getRowClosedSupplier()	RowsClosedSupplier
getScore()	UserScore
getRows()	Map<DiceColor, Row>
getRow(DiceColor)	Row
getRemainingMisses()	int
crossMiss()	void
crossField(DicePair)	void

Um ein Feld auf diesem Spielfeld nun anzukreuzen, muss die Methode *crossField(DicePair)* aufgerufen werden. Diese prüft das gegebene Würfelpaar und kreuzt dieses in der entsprechenden Reihe an.

Um einen Fehlwurf zu benutzen, muss die Methode *crossMiss()* verwendet werden.

Abseits der Methoden gibt es noch folgende Konstanten:

MIN_FIELDS_CROSSED_TO_CLOSE_ROW – Anzahl der Felder, die mindestens angekreuzt sein müssen, damit eine Reihe abgeschlossen werden kann.

AMOUNT_MISSES – Anzahl der Fehlwürfe für dieses Spielfeld.

Row	
ASC_FIRST_VALUE	int
ASC_LAST_VALUE	int
DESC_FIRST_VALUE	int
DESC_LAST_VALUE	int
Row(DiceColor, Order)	
getColor()	DiceColor
getOrder()	Order
getFirstValue()	int
getLastValue()	int
getFields()	List<Field>

1.5.3.1.1 Row

Eine Reihe hat eine festgelegte Farbe und eine Zahlenreihenfolge (Order). Diese Reihenfolge bestimmt, ob die Werte der Felder dieser Reihe auf- oder absteigend sortiert sind.

Falls diese Reihe aufsteigend (ASC) sortiert ist, so besitzt das erste Feld den Wert der Konstante *ASC_FIRST_VALUE* und das letzte Feld den Wert von *ASC_LAST_VALUE*.

Falls diese Reihe absteigend (DESC) sortiert ist, so werden die Werte der Konstanten *DESC_FIRST_VALUE* und *DESC_LAST_VALUE* verwendet.

1.5.3.1.1.1 Fields

Ein Feld hat einen festgelegten Wert und kann angekreuzt werden. Das Interface *Field* stellt folgende Methoden hierfür bereit:

Field	
getValue()	int
setCrossed(boolean)	void
isCrossed()	boolean

getValue() – liefert den Wert des Feldes

setCrossed(boolean) – der gegebene boolean gibt an, ob dieses Feld angekreuzt ist oder nicht.

isCrossed() – liefert boolean, ob dieses Feld angekreuzt ist.

AbstractField	
AbstractField(int)	
setCrossed(boolean)	void
getValue()	int
isCrossed()	boolean

1.5.3.1.1.2 AbstractField

AbstractField implementiert alle Methoden des Interfaces *Field*.

1.5.3.1.1.3 NumberField

Die Klasse *NumberField* erbt von *AbstractField* und stellt ein normales Zahlenfeld dar.

RowEndField	
ROW_END_FIELD_VALUE	int
RowEndField()	

NumberField	
NumberField(int)	

1.5.3.1.2 RowEndField

Die Klasse *RowEndField* erbt von *AbstractField* und stellt das letzte Feld in einer Reihe dar. Es wird bei Abschluss einer

Reihe angekreuzt und ist somit ein Zusatzfeld. Es hat einen festen Wert, der in der Konstante `ROW_END_FIELD_VALUE` festgelegt ist.

1.5.3.1.3 UserScore

Der `UserScore` repräsentiert den Punktestand eines Spielers (bzw. des Spielfeldes). Es werden die Punkte der einzelnen Reihen und der Fehlwürfe gespeichert.

UserScore	
SCORE_PER_MISS	int
SCORE_PER_CROSS	List<Integer>
UserScore()	
setScoreOfRow(DiceColor, int)	void
setScoreMisses(int)	void
getScoreOfRow(DiceColor)	int
getScoreMisses()	int
getScoreComplete()	int

setScoreOfRow() – Setzt für die gegebene Reihenfarbe die entsprechende Punktzahl. Die Punktzahl wird mittels der Liste *SCORE_PER_CROSS* ermittelt.

setScoreMisses() – Setzt den Punktestand für die Anzahl der gegebenen Fehlwürfe (Anzahl der gegebenen Fehlwürfe * *SCORE_PER_MISS*).

getCompleteScore() – liefert die Gesamtpunktzahl.

SCORE_PER_MISS – Punktzahl für einen Fehlwurf (Standard: -5 Punkte)

1.5.4 Das Spiel (Game)

RowClosedSupplier	
closeRow(DiceColor)	void
isRowClosed(DiceColor)	boolean

1.5.4.1 RowClosedSupplier

Das Interface `RowClosedSupplier` beinhaltet zwei Methoden.

closeRow(DiceColor) – die angegebene Reihe (Farbe) wird abgeschlossen.

isRowClosed(DiceColor) – gibt an, ob die gegebene Reihe abgeschlossen ist.

Game	
Game(IPlayer...)	
addPlayer(IPlayer...)	void
removePlayer(IPlayer...)	void
addDiceListener(DiceListener)	void
addGameListener(GameListener)	void
getDices()	List<IDice>
getPlayer()	List<IPlayer>
isPlaying()	boolean
startGame()	void
stopGame()	void
getWinningPlayer()	List<IPlayer>
closeRow(DiceColor)	void
isRowClosed(DiceColor)	boolean

1.5.4.2 Game

Diese Klasse repräsentiert ein Spiel. Hierfür stellt diese Klasse verschiedene Methoden zur Verfügung. Die wichtigsten werden im Folgenden erläutert:

addPlayer(...) – Fügt die gegebenen Spieler diesem Spiel hinzu.

startGame() – Startet dieses Spiel in einem neuen Thread.

stopGame() – Stoppt dieses Spiel (und den dazugehörigen Spiel-Thread).

getWinningPlayer() – Liefert eine Liste mit allen Spielern, die dieses Spiel gewonnen haben (mehrere bei Punktgleichstand).

Die Klasse `Game` implementiert zudem das Interface `RowClosedSupplier`.

2 Signifikante Implementierungsdetails

2.1 Abschließen von Reihen

Reihen werden in einem Spiel nicht unmittelbar nach dem Ankreuzen der letzten Zahl geschlossen, sondern erst nachdem alle Spieler ihre Aktion ausgeführt haben (ein Würfelpaar ausgewählt und angekreuzt haben). Das hat den einfachen Grund, dass mehrere Spieler gleichzeitig die gleiche Farbreihe abschließen dürfen.

```
private Set<DiceColor> closedRows = new HashSet<>();
private Queue<DiceColor> rowsToCloseAfterRoundFinished = new ArrayDeque<>();

@Override
public void closeRow(DiceColor color) {
    System.out.println("Reihe wird nach Abschluss des Zuges geschlossen: " + color);
    rowsToCloseAfterRoundFinished.add(color);
}
```

Diese Anforderung wurde mittels einer *Queue* realisiert, in die die zu schließenden Reihen eingefügt werden.

```
private void closeQueuedRows() {
    while (!rowsToCloseAfterRoundFinished.isEmpty()) {
        System.out.println("Row is now closed: " + rowsToCloseAfterRoundFinished.peek());
        closedRows.add(rowsToCloseAfterRoundFinished.poll());
    }
}
```

Nachdem alle Spieler ihre Aktion ausgeführt haben, werden alle Reihen aus der *Queue* *rowsToCloseAfterRoundFinished* entfernt und in die Liste *closedRows* hinzugefügt.

2.2 GameThread

Das Spiel läuft in einem eigenen *Thread*. Dieser Thread wird in der Klasse *Game* über die Methode *createGameThread()* erzeugt. Die Variable *isPlaying* gibt an, ob das Spiel läuft.

Zu Beginn des *Threads* wird die Variable *isPlaying* auf *true* gesetzt, da das Spiel mit Start dieses *Threads* beginnt. Danach folgt die Spieleschleife, die solange läuft, bis *isPlaying* auf *false* gesetzt wird.

```
private void createGameThread() {
    gameThread = new Thread(() -> {
        isPlaying = true;

        try {
            // Spieleschleife
            while (isPlaying) {

                // Jeder Spieler kommt pro Durchgang 1x dran
                for (IPlayer currentPlayer : this.players) {

                    System.out.println("\n\nSpieler ist an der Reihe: " + currentPlayer);
                    gameListeners.forEach(l -> l.nextPlayersTurn(currentPlayer));

                    // Würfel würfeln
                    rollDice();
                    List<DicePair> whiteDices = getWhiteDicePairs(); // Aus den weißen Würfel Würfelpaare bilden
                    List<DicePair> colorDices = getColorDicePairs(); // Aus den Farbwürfeln Würfelpaare bilden

                    // Andere Spieler wählen das weiße Würfelpaar aus
                    System.out.println("----- Alle anderen Spieler wählen weiße Würfel");
                    letOtherPlayerChooseWhiteDices(whiteDices, currentPlayer);

                    // Der aktive Spieler wählt nun aus den weißen und den Farbwürfeln Würfelpaare aus
                    System.out.println("----- Aktiver Spieler wählt Würfelpaare");
                    letPlayerSelectDice(currentPlayer, whiteDices, colorDices);

                    closeQueuedRows();
                    if (isGameOver()) {
                        isPlaying = false;
                        break;
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Game over");
        gameListeners.forEach(l -> l.gameOver(this));
    }, "Game"); // Thread mit Namen 'Game' erstellen
}
```

In jedem Spieleschleifendurchlauf und somit in jeder Spielrunde ist jeder Mitspieler einmal der aktive Spieler. Der Zug des aktiven Spielers hat folgenden Ablauf:

1. Die Würfel werden geworfen

2. Es werden weiße- und Farbwürfelpaare gebildet
3. Alle nicht-aktiven Spieler wählen ein Würfelpaar aus den weißen Würfelpaaren
4. Der aktive Spieler ist an der Reihe
 - a. Auswahl eines Würfelpaares aus den weißen Würfeln
 - b. Es wird geprüft, ob das Spiel vorbei ist
 - c. Auswahl eines Würfelpaares aus den Farbwürfeln
5. Es wird geprüft, ob das Spiel vorbei ist

```
private void letPlayerSelectDice(IPlayer player, List<DicePair> whiteDices, List<DicePair> colorDices) {
    DicePair selectedWhiteDice = letPlayerSelectWhiteDice(player, whiteDices);

    // Falls der Benutzer einen Fehlwurf angekreuzt hat, so soll der keinen
    // Farbwürfel mehr wählen dürfen.
    if (DicePair.MISS.equals(selectedWhiteDice)) {
        // User crossed MissField
        // prevent user to cross second dice
        player.getGameBoard().crossMiss();
        return;
    }

    closeQueuedRows();
    if (isGameOver()) {
        isPlaying = false;
        return;
    }

    DicePair selectedColorDice = letPlayerSelectColorDice(player, colorDices);

    // Prüfen, ob der Player 2x keinen Würfel ausgewählt hat bzw.
    // ob der Spieler beim ersten Mal keinen Würfel gewählt hat und beim zweiten Mal
    // dann ein Fehlwurf angekreuzt hat
    // -> Dies ist gleichbedeutend mit einem Fehlwurf
    if (isEmpty(selectedWhiteDice) && isEmptyOrMiss(selectedColorDice)) {
        // Fehlwurf ankreuzen
        player.getGameBoard().crossMiss();
    }
}
```

Bei der Auswahl von Würfelpaaren des aktiven Spielers wählt dieser zuerst ein weißes Würfelpaar. Anschließend werden alle eingereichten Reihen geschlossen und es erfolgt eine Prüfung, ob eine Spielendbedingung erfüllt ist; das Spiel wird dann ggf. beendet. Ansonsten wählt der aktive Spieler das zweite und somit das Farbwürfelpaar. Falls der Spieler bei beiden Würfelpaaren kein Paar bzw. beim zweiten einen Fehlwurf ausgewählt hat, so wird ein Fehlwurf auf dem Spielfeld des Spielers eingetragen.

2.2.1 Auswahl eines Würfels

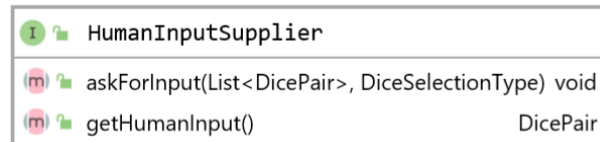
Die Auswahl und das Ankreuzen eines gewählten Würfels wird in der Methode *letPlayerSelectDice(...)* gehandhabt. Sobald der Spieler ein Würfelpaar ausgewählt hat und dieses kein leeres Würfelpaar oder ein Fehlwurf ist, soll dieses Würfelpaar angekreuzt werden. Falls beim Ankreuzen und der damit verbundenen Überprüfung auf Richtigkeit des gewählten Würfelpaares ein Fehler auftritt und eine *Exception* geworfen wird, wird diese abgefangen und der Spieler wird erneut aufgefordert ein Würfelpaar zu wählen. Dies geschieht so lange, bis der Spieler ein gültiges Würfelpaar ausgewählt hat.

```
private DicePair letPlayerSelectDice(BiFunction<IPlayer, List<DicePair>, DicePair> diceSelection, IPlayer player,
    List<DicePair> dices) {
    // Der Spieler wählt so lange einen Würfel aus den gegebenen Würfel aus, bis er
    // einen gültigen Würfel ausgewählt hat.
    while (true) {
        try {
            // Spieler wählt aus den gegebenen Würfelpaaren ein Würfelpaar aus
            DicePair selectedDice = diceSelection.apply(player, dices);
            if (!isEmptyOrMiss(selectedDice)) {
                // Den Wurf nur ankreuzen, falls ein Würfel ausgewählt wurde
                crossSelectedDice(player, selectedDice, dices);
            }

            // Wenn bis hierher keine Fehlermeldung kam, dann hat der Spieler ein gültiges
            // Würfelpaar ausgewählt, dieses Würfelpaar wird zurückgegeben.
            return selectedDice;
        } catch (IllegalArgumentException e) {
            System.out
                .println("Der Spieler " + player + " hat keinen gültigen Würfel ausgewählt: " + e.getMessage());
            gameListeners.forEach(l -> l.invalidDiceChoiceMade(player, e.getMessage()));
        }
    }
}
```

2.3 Auf Eingabe von Benutzer warten

Das Warten auf die Eingabe des Benutzers stellte zunächst ein Problem dar, denn das Model ist von dem Benutzerinterface entkoppelt. Es musste somit ein Weg gefunden werden, dass das Model auf Eingaben von dem Benutzer wartet und erst dann weitere Aktionen ausführt.



Gelöst wurde dieses Problem mittels eines Interfaces, dem `HumanInputSupplier`. Dieses Interface bietet Methoden zum Erfragen und Holen einer Benutzereingabe. Das Model erfragt zuerst über die Methode `askForInput(...)` eine Eingabe des Benutzers und wartet anschließend mittels `wait()` auf das Ergebnis, welches über `getHumanInput()` geholt wird.

```
private DicePair waitForSelectionAndCrossField(List<DicePair> dices, DiceSelectionType selectionType) {
    DicePair selection = null;
    synchronized (this) {
        try {
            System.out.println("Benutzer nach Auswahl des Würfelpaares fragen: " + dices);
            inputSupplier.askForInput(dices, selectionType);

            // Auf Eingabe des Benutzers über die UI warten
            while ((selection = inputSupplier.getHumanInput()) == null) {
                this.wait();
            }
        } catch (InterruptedException e) {
            System.err.println("Der Spieler hat das Spiel verlassen oder "
                + "das System hat diesen Thread unterbrochen! " + e.getMessage());
        }
    }
    return selection;
}
```







Die UI registriert sich beim Spieler als `HumanInputSupplier` und implementiert beide Methoden. Sobald der Spieler eine Auswahl getroffen hat, wird die Methode `playerSelectedDice(...)` aufgerufen.

```
private void playerSelectedDice(DicePair dice) {
    try {
        humanInput = dice;
        disableAllButtons();
    } finally {
        synchronized (player) {
            player.notify();
        }
    }
}
```



Diese Methode speichert die Auswahl in der Variable `humanInput`, damit diese später über die Methode `getHumanInput()` geholt werden kann. Anschließend benachrichtigt es das Model (den Spieler) mittels `notify()` darüber, dass der Spieler eine Auswahl getroffen hat.

2.4 Darstellen eines Würfels

Die Würfelaugen werden mittels einer *switch-case-Anweisung* angezeigt bzw. sichtbar gemacht. Um redundanten Code zu vermeiden, wurden die Würfelaugen in zwei Gruppen eingeteilt.

Gruppe 1	Gruppe 2
  	  

Diese Unterteilung erfolgte nach folgendem Prinzip:

Die 1 ist in der 3 enthalten; die 3 ist in der 5 enthalten	
Die 2 ist in der 4 enthalten; die 4 ist in der 6 enthalten	

Somit lässt sich

- die 5 durch eine 3 und zwei weiteren Punkten darstellen
- die 3 durch eine 1 und zwei weiteren Punkten darstellen
- die 6 durch eine 4 und zwei weiteren Punkten darstellen
- die 4 durch eine 2 und zwei weiteren Punkten darstellen

Nach diesem Prinzip ist auch die *switch-case-Anweisung* aufgebaut.

```
public void refreshDice() {
    hideDiceEyes();

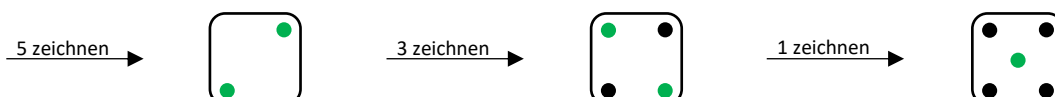
    // 1: _ 2: _ 3: _ 4: _ 5: _ 6: _
    // - - # - - # - - # - - # - - # - - #
    // - # - - - # - - - - - # - - - - - #
    // - - - - - # - - - - - # - - - - - #
    switch (dice.getCurrentValue()) {
        case 5:
            // für die 5 wird auch die 3 und die 1 gebaut
            diceEyes[0][2].setVisible(true); // oben rechts
            diceEyes[2][0].setVisible(true); // unten links
        case 3:
            // für die 3 wird auch die 1 gebaut
            diceEyes[0][0].setVisible(true); // oben links
            diceEyes[2][2].setVisible(true); // unten rechts
        case 1:
            diceEyes[1][1].setVisible(true); // mitte mitte
            break;
        case 6:
            // für die 6 wird auch die 4 und die 2 gebaut
            diceEyes[0][1].setVisible(true); // links mitte
            diceEyes[2][1].setVisible(true); // rechts mitte
        case 4:
            // für die 4 wird auch die 2 gebaut
            diceEyes[0][2].setVisible(true); // oben rechts
            diceEyes[2][0].setVisible(true); // unten links
        case 2:
            diceEyes[0][0].setVisible(true); // oben links
            diceEyes[2][2].setVisible(true); // unten rechts
            break;
        default:
            throw new IllegalArgumentException("Zahl von Würfel nicht bekannt: " + dice.getCurrentValue());
    }
}
```

Bei einer 5 werden zwei Augen (oben rechts, unten links) und die 3 gezeichnet. Die 3 zeichnet zwei Augen (oben links, unten rechts) und die 1. Die 1 zeichnet dann lediglich das mittlere Würfelauge und bricht aus der *switch-case-Anweisung* mittels *break* aus.

Die 6, 4 und 2 werden nach dem gleichen Schema gezeichnet.

Beispielhafte Darstellung des Zeichenvorgangs einer 5:

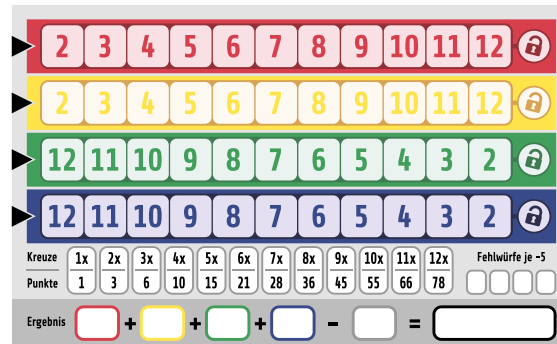
● = neu gezeichnetes Würfelauge



3 Das User Interface (UI)

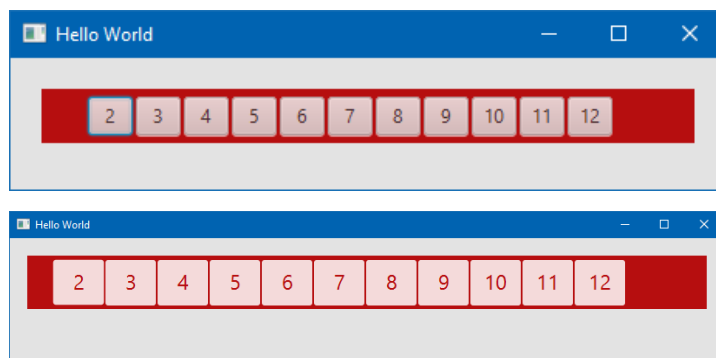
3.1 Aufbau des Spielfeldes

Das Ziel beim Aufbau des Spielfeldes ist es, den originalen Qwixx Spielblock möglichst genau nachzubilden. Der Spielblock ist nachfolgend abgebildet.

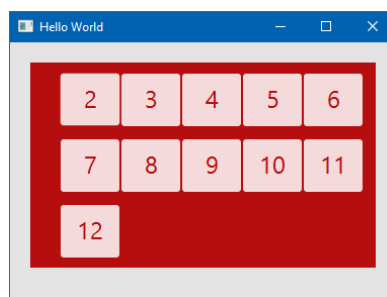


https://commons.wikimedia.org/wiki/File:Qwixx_scorecard_nofonts.svg

Der Aufbau des Spielfeldes begann mit der Gestaltung der Farbreihen. Diese wurden zunächst mittels eines *TilePanels* erstellt, in das 11 *Buttons* mit den Zahlenwerten 2 bis 12 (oder in umgekehrter Reihenfolge) hinzugefügt worden sind. Diese wurden noch mittels css optisch angepasst.



Es wurde ein *TilePane* verwendet, da das Verhalten zum Wrappen von den beinhalteten Komponenten beim Anpassen der Größe als nützlich erachtet wurde.



Wie sich im späteren Verlauf gezeigt hat, ist diese Eigenschaft allerdings nicht mehr von Relevanz, da das Spielfeld eine Mindestgröße erhalten hat und somit nicht so weit verkleinert werden kann, sodass das Verhalten des *TilePanels* nicht benötigt wird.

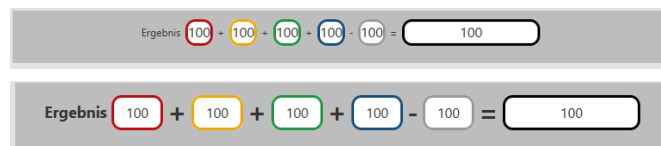
Als nächstes wurden die *Buttons* der Reihe weiter modifiziert. Beim Anklicken eines *Buttons* erscheint nun über diesem ein schwarzes Kreuz mittels eines *ImageViews*. Zudem wird dieses Kreuz über die Eigenschaft *opacity* in abgeschwächter Form auch beim *Hovern* über dem *Button* angezeigt. Der Benutzer erhält somit ein optisches Feedback.



Als nächstes wurde das Bonusfeld mit Hilfe eines *ImageViews* von einem Schloss gestaltet, anschließend wurde der schwarze Pfeil am linken Rand über ein *Triangle* eingefügt.



Die Reihen waren damit fertig gestaltet. Nun fehlte noch der Punktestand am Ende des Spielblocks und darüber die Punktelegende mit den Fehlwurffeldern. Zuerst wurde mit der Gestaltung des Punktestandes begonnen. Hierfür wurden ausschließlich *Labels* verwendet, die noch passend gestyled wurden.



Beim Erstellen und Anordnen der Punktzahllgende wurde zunächst an ein *GridPane* gedacht. Hiermit hätte sich allerdings die Darstellung der einzelnen Punkte-Pro-Kreuz-Kästen schwierig gestaltet. Daher wurde eine neue Klasse *BreakLabel* entworfen (grün umrandet). Diese Klasse stellt nun zwei *Labels* übereinander dar, getrennt von einem *Separator*.

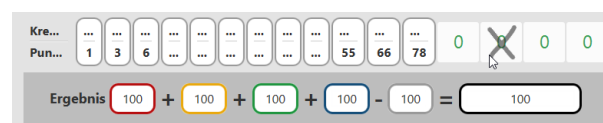


Diese *Labels* wurden dann mit der Anzahl der Kreuze und der dazugehörigen Punktzahl befüllt. Das *BreakLabel* hat noch über *css* einen passenden Style bekommen, damit dieses optisch der Vorlage entspricht.

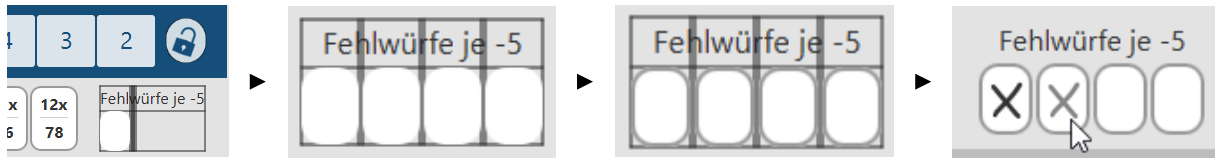
Kreuz	1x	2x	3x	4x	5x	6x	7x	8x	9x	10x	11x	12x
Punkte	1	3	6	10	15	21	28	36	45	55	66	78

Die vorangestellte Beschreibung $\frac{\text{Kreuz}}{\text{Punkte}}$ wurde ebenfalls mit einem *BreakLabel* realisiert, wobei hier im Nachhinein noch der *css-Style* entfernt wurde.

Nun fehlte nur noch die Anzeige der Fehlwürfe. Diese wurden testweise durch vier *Buttons* realisiert.



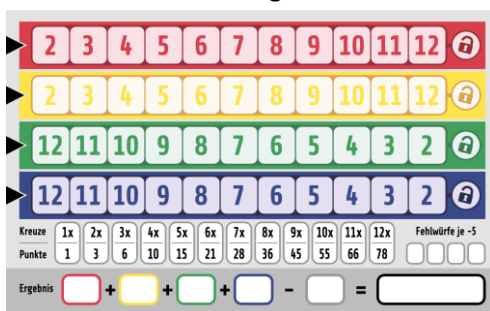
Aus diesen *Buttons* wurde dann die Klasse *MissField* entwickelt, welche ebenfalls ein schwarzes Kreuz beim Anklicken und *Hovern* darstellt.



Die *MissFields* wurden dann mittels eines *GridPanels* angeordnet. Oberhalb wurde noch ein beschreibender Text eingefügt.

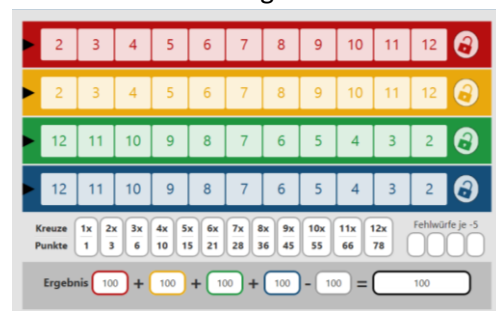
Damit war das Spielfeld fertig. Optisch ähnelt es stark der Vorlage, was erreicht werden sollte. Die Farben wurden etwas kräftiger gewählt, damit diese nicht zu grell werden.

Vorlage



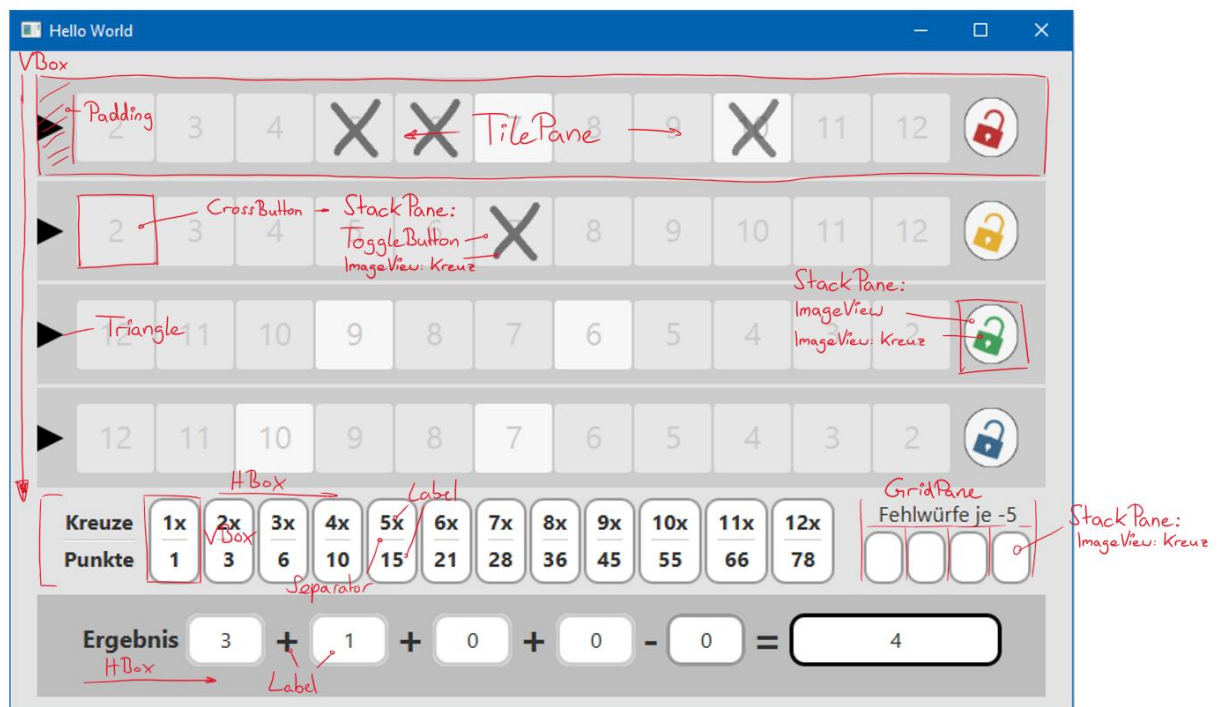
https://commons.wikimedia.org/wiki/File:Qwixx_scorecard_nofonts.svg

Nachbildung in JavaFX



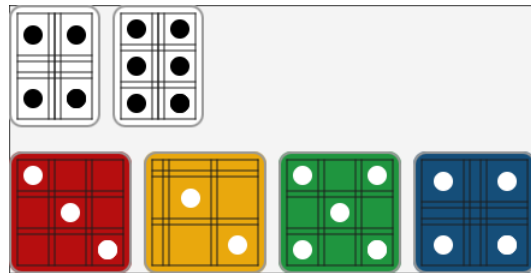
3.2 Der gesamte Spielblock im Detail

Im Nachfolgenden ist der komplett gestaltete Spielblock mitsamt der verwendeten grafischen Elemente aufgezeigt. Die Farben der Reihen wurden der Übersichtlichkeit halber entfernt.



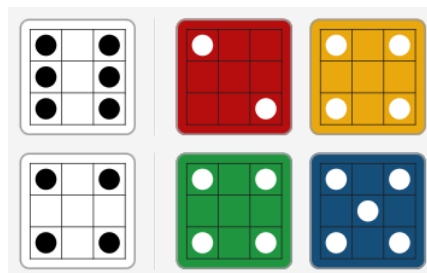
3.3 Darstellung der Würfel

Die Würfel sollten mittels *Circles* auf einem 3x3 *GridPane* dargestellt werden. Die erste Version hatte allerdings die Eigenschaft, dass die Würfel nicht immer quadratisch und auch die Würfelaugen nicht zentriert waren.



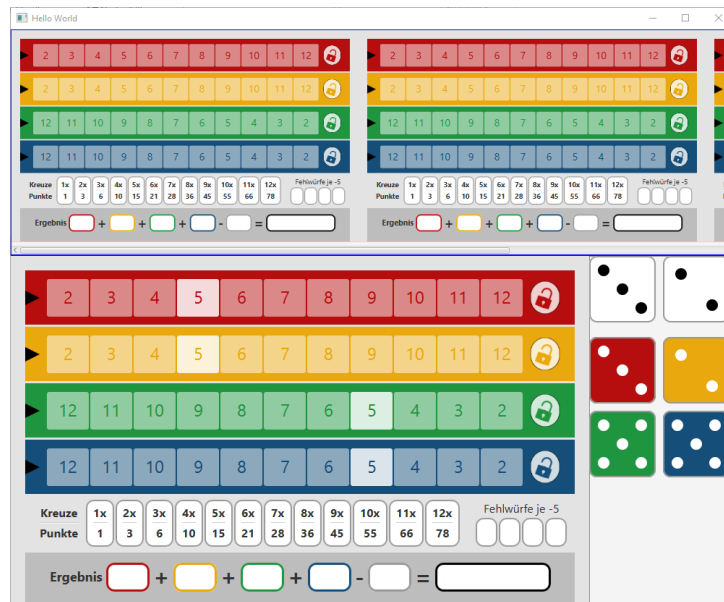
Dieses Verhalten war der Tatsache geschuldet, dass das *GridPane* leere Zellen kleiner darstellt als befüllte Zellen. So verschoben sich die Würfelaugen und die Würfel wurden z.T. rechteckig.

Dieses Verhalten wurde gelöst, indem der Würfel bzw. das *GridPane* des Würfels konstant 9 Würfelaugen besitzt, diese allerdings teilweise ausgeblendet werden. So beinhaltet jede Zelle des *GridPanes* immer ein Element und wird nicht kleiner skaliert als andere Zellen.



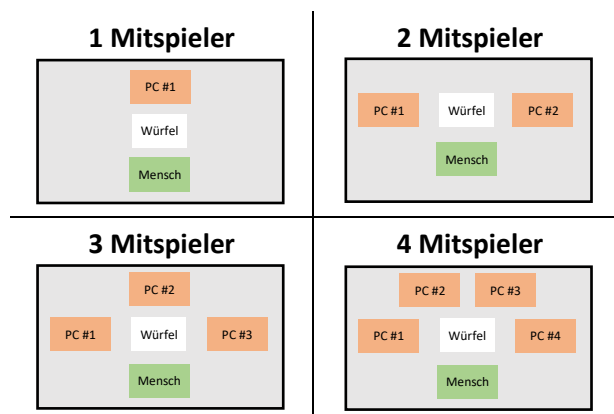
3.4 Anordnung mehrerer Spielfelder

Ein Spielblock allein macht kein fertiges Spiel aus. Hierfür benötigt es Mitspieler und somit auch mehrere Spielblöcke. Die Anordnung dieser Spielblöcke der Mitspieler auf einem großen Spielfeld war nun das Problem. Der erste Lösungsversuch sah vor, dass der Spielblock des menschlichen Spielers am unteren Rand dargestellt wird. Die Spielblöcke der Mitspieler sollten verkleinert in einem *ScrollPane* über dem Spielblock des menschlichen Spielers dargestellt werden. Die Würfel wurden am rechten Rand positioniert.



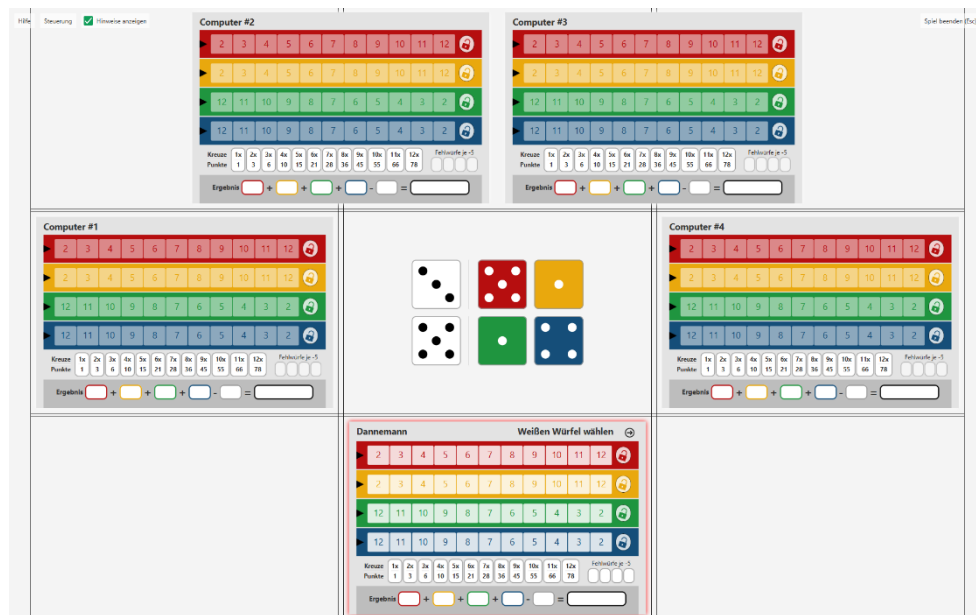
Dies sah optisch allerdings nicht ansprechend aus und bei mehreren Mitspielern wurden so teilweise die Spielblöcke der Mitspieler verdeckt bzw. waren außerhalb des Spielfensters.

Aus diesem Grund wurde die Anordnung der Spielblöcke an jener auf einem Tisch orientiert. Dort liegen zumeist die Würfel in der Mitte und die Spielblöcke sind in einem Kreis um die Würfel verteilt. Dies ergab bei unterschiedlich vielen Mitspielern folgendes Layout:



Dieses Layout kann mittels eines 3x3 *GridPanes* erreicht werden. Die Würfel befinden sich stets an der Position (1, 1) und der Spielblock des menschlichen Spielers auf (1, 2). Die Positionen der Spielblöcke der Mitspieler variieren je nach der Anzahl der Spieler.

Konkret sieht dies im Fall von 4 Mitspielern wie folgt aus:



4 JUnit-Tests

Für die Model-Klassen (wie z.B. *Player*, *GameBoard* etc.) sind JUnit-Tests zur Absicherung deren korrekten Funktionalität erstellt worden. Diese Tests decken mindestens 80% des geschriebenen Quellcodes der Klassen ab.

5 Verwendete Bibliotheken

In diesem Projekt wurde die Bibliothek *JFoenix* verwendet, um grafische Elemente im *Material Design* darzustellen. Dies verleiht dem Programm ohne viel Aufwand ein modernes und dem Benutzer vertrautes Aussehen.

Zusammenfassung

In dieser Projektarbeit wurde das Spiel „Qwixx“ als digitales Spiel in Java entwickelt. Es wurden Klassen zur Abbildung des Spieles erarbeitet und umgesetzt. Zur Qualitätssicherung wurden für die meisten dieser Klassen JUnit-Tests geschrieben. Die Oberfläche wurde optisch nach der Vorlage des Spiels gestaltet.

Abschließend kann man sagen, dass im Rahmen dieser Projektarbeit eine spielbare digitale Version des Gesellschaftsspiels „Qwixx“ entwickelt worden ist, die noch einige Erweiterungsmöglichkeiten bietet.

6 Erweiterung

6.1 Funktionale Erweiterung

Als Erweiterungsmöglichkeit können noch Statistiken in Verbindung einer Datenbank eingebracht werden. Hier kann jedes Spiel eines Spielers mit der jeweiligen Anzahl der Mitspieler und dem Gewinner abgespeichert werden. Anhand dieser gesammelten Daten können zudem anschauliche Grafiken und Statistiken (wie z.B. einen Highscore) dem Benutzer angezeigt werden.

Dies kann z.B. über die vollständig in Java programmierte *HyperSQL* Datenbank (*Hsqldb*) realisiert werden, welche zudem frei verfügbar ist. Ein weiterer Vorteil der *Hsqldb* ist, dass diese als einzelne JAR direkt mit dem Programm mitgeliefert werden kann und keine weitere Installation erfordert.

Eine weitere Erweiterungs- bzw. Verbesserungsmöglichkeit bietet das Schreiben von mehr JUnit-Tests. Diese decken zzt. nur den Großteil der Model-Klassen ab.

6.2 Grafische Erweiterung

Zudem gibt es auch Erweiterungsmöglichkeiten in der grafischen Benutzeroberfläche.

Es können z.B. noch Würfelanimationen hinzugefügt werden. Bei jedem neuen Wurf der Würfel wird diese Animation abgespielt, sodass der Benutzer einen Übergang von einem Wurf zum Nächsten erkennen kann und sich nicht nur schlagartig die Augenzahl auf den Würfeln ändern.

Des Weiteren kann ein Delay beim Ankreuzen von Feldern der Computergegner eingebaut werden, um den Denkprozess der Mitspieler zu simulieren. Die Vorgehensweise sähe z.B. wie folgt aus:

1. Die Klasse *ComputerGameBoard* überschreibt die Methode *fieldCrossed(...)*
2. Es wird in einem neuen *Thread* eine zufällige Zeit gewartet (um das Ankreuzen zu verzögern)
3. Nach dieser Wartezeit wird die Methode (und somit das Ankreuzen) fortgesetzt mittels des Aufrufs *super.fieldCrossed(...)*

```
@Override
public void fieldCrossed(Row rowToCross, Field fieldToCross) {
    new Thread(() -> {
        try {
            Thread.sleep(new Random().nextInt(700) + 200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        super.fieldCrossed(rowToCross, fieldToCross);
    }).start();
}
```

7 Quellenverzeichnis

Spieleanleitung <https://www.brettspiele-report.de/images/q/qwixx/Spielanleitung-Qwixx.pdf>

JFoenix <http://www.jfoenix.com/>
<https://github.com/jfoenixadmin/JFoenix>