Entwurf der schriftlichen Arbeit zum Jugend-forscht-Projekt „Ein Computerprogramm zur Unterstützung des mentalen Trainings der in einem Sportverein des Deutschen Schützenbundes e. V. organisierten jugendlichen Sportschützen“

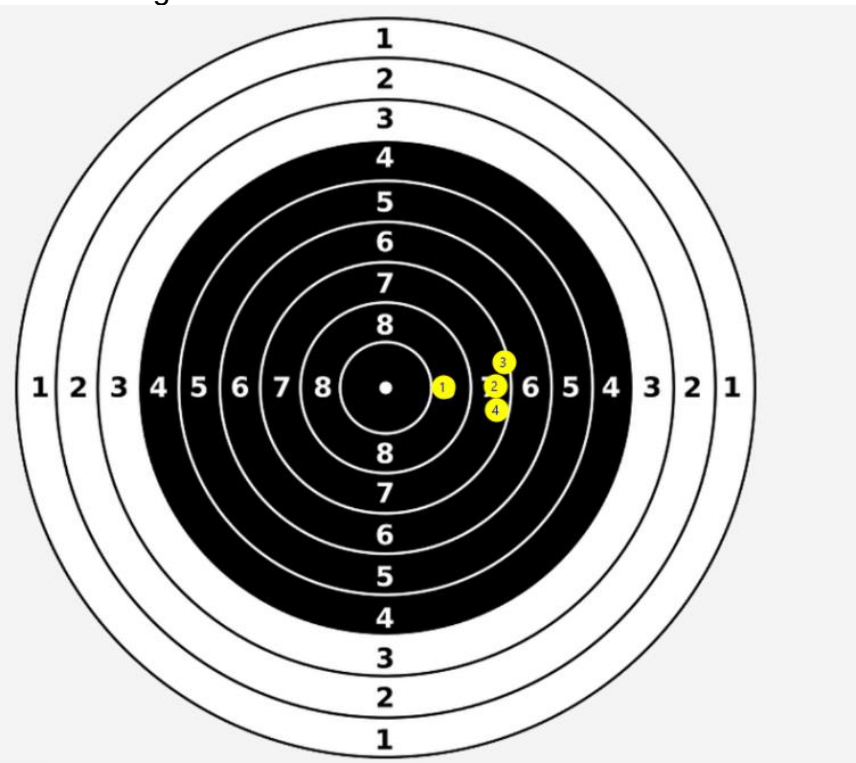
Gliederung

1. Einleitung
    a. Beschreibung der Problematik und des prinzipiellen Lösungsansatzes der Arbeit
    b. Zusammenfassung des aktuellen technischen Stands dieser Thematik
2. Dokumentation der Entwicklung des Programms
    a. Begründung der Nutzung der verwendeten Programmiersprache JavaFX
    b. Dokumentation des Hauptalgorithmus
        i. Grundidee
        ii. Teil 1: nötige Festlegungen
        iii. Teil 2: die grundlegende Arbeitsweise des Algorithmus
        iv. Teil 3: die Generierung der Empfehlung
    c. Umsetzung in JavaFX
        i. Umsetzung der grafischen Oberfläche und der Ein- und Ausgabe
        ii. Implementierung weiterer, wichtiger Algorithmen
            1. Umsetzen der Eingabe eines Schusswerts
            2. Umsetzen der Protokollerstellung
            3. Umsetzen der Zeitfunktionen
    d. Beschreibung des Arbeitsprozesses
3. Bewertung des Programms
    a. Auswertung der Tests des fertigen Programms
    b. Vergleich des fertigen Programms mit seinen Zielen
4. Literaturverzeichnis
5. Anhang
    a. Der Quelltext des Programms
    b. Liste der Variablen im Hauptalgorithmus und derer Funktion
    c. ausführbare Version des Programms als .jar-Datei
    d. Testprotokolle von Programm und Algorithmus in Bild und Text

1. Einleitung

a. Beschreibung der Problematik und des prinzipiellen Lösungsansatzes der Arbeit

- Wettkampf → schwierige Situation für Sportschützen
- einerseits: körperliche/ sportliche Herausforderungen (Stand, Anlegen der Waffe, schließlich auch das Treffen des Ziels)
- andererseits: auch geistige Komponente
- Beispiel: Ziel sind 88 Ringe bei 10 Schuss
- Schütze schießt folgendermaßen:



- ist Erreichen des Ziels noch möglich?
- 88 Ringe, 10 Schuss → 8,8 Ringe pro Schuss im Durchschnitt sind Ziel
- Rechnung: 9 ist Optimum → 3 mal 2 daneben, also theoretisch 3 mal 11 zum Ausgleich, aber das geht hier nicht, also neuen Durchschnitt bilden:
- $Optimum = \frac{88 - (3*7+1*8)}{6} = \frac{59}{6} \approx 10$ → noch zu schießende Ringe durch übrige Schüsse
- → möglichst jeder Schuss eine 10
- was braucht man mindestens?
- $Minimum = 88 - (3 * 7 + 1 * 8) - 5 * 10 = 9$ → noch zu schießende Ringe minus mit den übernächsten Schüssen höchstens zu erreichende Punktzahl
- → Empfehlung wäre hier: nächster Schuss mindestens eine 9, Versuch, 10 zu erreichen, hier sinnvoll
- diese Rechnungen ohne Taschenrechner immer wieder unter Erfolgs- und Zeitdruck
- → enorme psychische Belastung; schwierig, das zu lernen, besonders in der hier betreffenden Altersklasse
- → Programm soll nicht als Hilfsmittel im Wettbewerb fungieren, sondern die Schützen im Training an genau die Situation gewöhnen, ihnen helfen, selbst die

entsprechenden Empfehlungen zu errechnen, zudem noch beim Zeitmanagement unterstützen sowie weitere nützliche Informationen (letzter Schuss, allgemeiner Trend, …) ausgeben → für Schützen
- Programm soll für verschiedene Waffen (Luftgewehr, Pistole…) und verschiedene Schussserien und Zielringzahlen nutzbar und mit Wettbewerbsformalismen (z.B. Zeitvorgaben) kompatibel sein
- für Trainer: Protokoll mit Auswertung generieren → gezieltes Erkennen von Schwachstellen des Schützen, die gezielt behoben werden können, Vereinfachung der Trainingsprotokollierung, Möglichkeit, Ergebnisse von heute mit denen vor einiger Zeit zu vergleichen
- einfach und unter Stress bedienbare, intuitive Ein- und Ausgabe

b. Zusammenfassung des aktuellen technischen Stands dieser Thematik

- nach Befragung des Experten Hr. Kunze: genau dieses Programm gibt es noch nicht
- kleiner, sehr spezialisierter Markt für das Programm; Programm großer Software-Schmiede würde sich für diesen nicht lohnen
- zudem: Programm ist auf praktische Anwendung z.B. im Verein des Tests so „maßgeschneidert“, dass es durch ein allgemeines Programm nicht ersetzt werden kann
- bisher: wirklich keine bekannte Hilfe für Sportschützen auch nur ähnlich hierzu; einzig ein automatisiertes Programm zur Ermittlung des Schussergebnisses von einer elektronischen Scheibe existiert
- *Anmerkung: erneut gründlich recherchieren*
- *Anmerkung: hier käme dann noch ein Interview mit dem Experten Hr. Kunze oder eine Beschreibung, wie man genau das, wobei das Programm unterstützt, früher gelernt hat*

2. Dokumentation der Entwicklung des Programms

a. Begründung der Nutzung von JavaFX als Programmiersprache

[1]

- JavaFX → Java-Version, welche besonders auf grafische Oberflächen ausgelegt ist
- eigentlich für leistungsstarke Grafiken und Videospiele entwickelt
- Trennung der eigentlichen Oberfläche (deren reiner Grafik mit Buttons, Textfeldern zur Eingabe,… und deren Eigenschaften) von deren Code (der z.B. beim Aufruf von Buttons ausgeführt wird) und dem Hauptprogramm (welches eigentlich nur zum Programmstart gebraucht wird) in drei Dateien (FXML-Oberflächenformatierung und zwei Java-Quellcodes) macht Bearbeitung einfacher und übersichtlicher
- moderneres Aussehen, mehr grafische Möglichkeiten als mit den alten Versionen Swing oder AWT
- Zukunftssicherheit von JavaFX (neueste Version der Sprache)
- hervorragende Integration von JavaFX in die Entwicklungsumgebung NetBeans IDE 8.0.2 sowie JavaFXSceneBuilder (ein Programm zur automatischen Erstellung von grafischen Oberflächen für JavaFX-Anwendungen) macht erleichtert Entwicklung

- allgemeine Java-Vorteile:
  - allgemein auf einfache Entwicklung ausgelegt
  - viele Möglichkeiten, die die Sprache intern schon mitbringt (Erledigung aller Aufgaben der grafischen Oberfläche ohne Java-Externe Klassen)
  - Beispiele in diesem Programm: z.B. Möglichkeit, in JavaFX ein Element der grafischen Oberfläche abzufotografieren und zu speichern, mit der Funktion Snapshot der Klasse Pane und WritableImages und Filewriters möglich; alles in Java enthalten
  - plattformübergreifende Nutzung (Beispiel: frühe, nicht-grafische C++-Variante des Programms musste für ein 32-Bit-Windows neu kompiliert werden, Java(FX) wird auf fast jedem Desktopbetriebssystem unterstützt- für Windows ab XP, Linux in allen verbreiteten Distributionen, MacOS, Solaris… verfügbar)
  - Verbreitung von Java (offizielle Angaben um (*aktuellen Wert einfügen!)* 6,5 Milliarden Geräte, auf denen es ausgeführt wird)
  - große Java-Entwicklergemeinde mit guter Aussicht, Hilfe in Foren oder der offiziellen Dokumentation zu finden
  - Kenntnis von JavaFX und Java-Swing aus dem Unterricht

[1] http://code.makery.ch/assets/library/javafx-2-tutorial/javafx-logo.png, 08.12.15

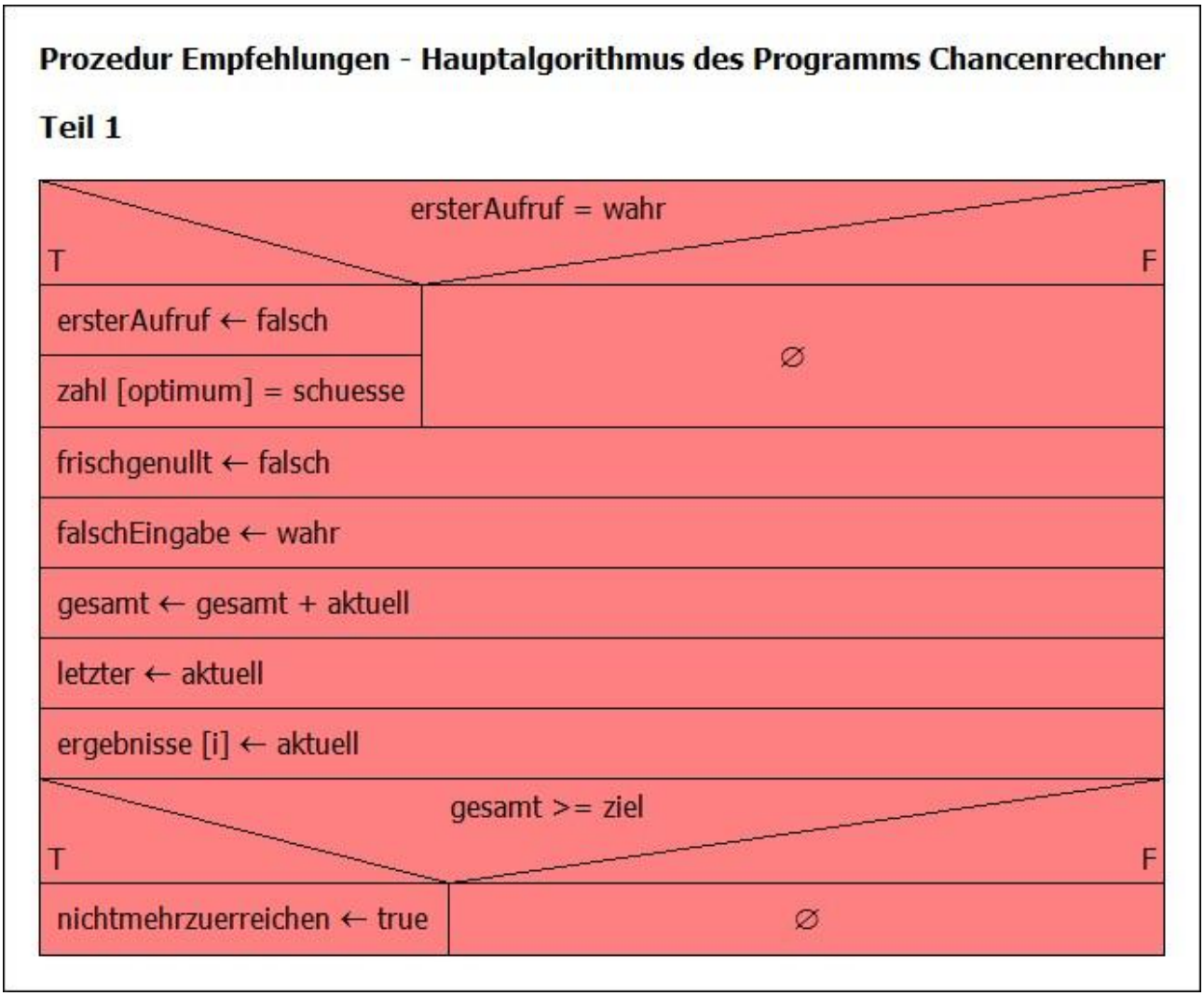
b. Dokumentation des Hauptalgorithmus

- folgend: Code in Struktogrammen dargestellt und erklärt
- Festlegung betreffend Farben:
  - lachsrot → im Programm nötige Festlegungen, besonders Variableninitialisierungen
  - gelb → Ausgaben
  - blau → Hauptalgorithmus → Fall 1: zu wenig geschossen
  - cyan → Hauptalgorithmus → Fall 2: zu viel geschossen
  - pink → Hauptalgorithmus: Standardaktion

i. Grundidee

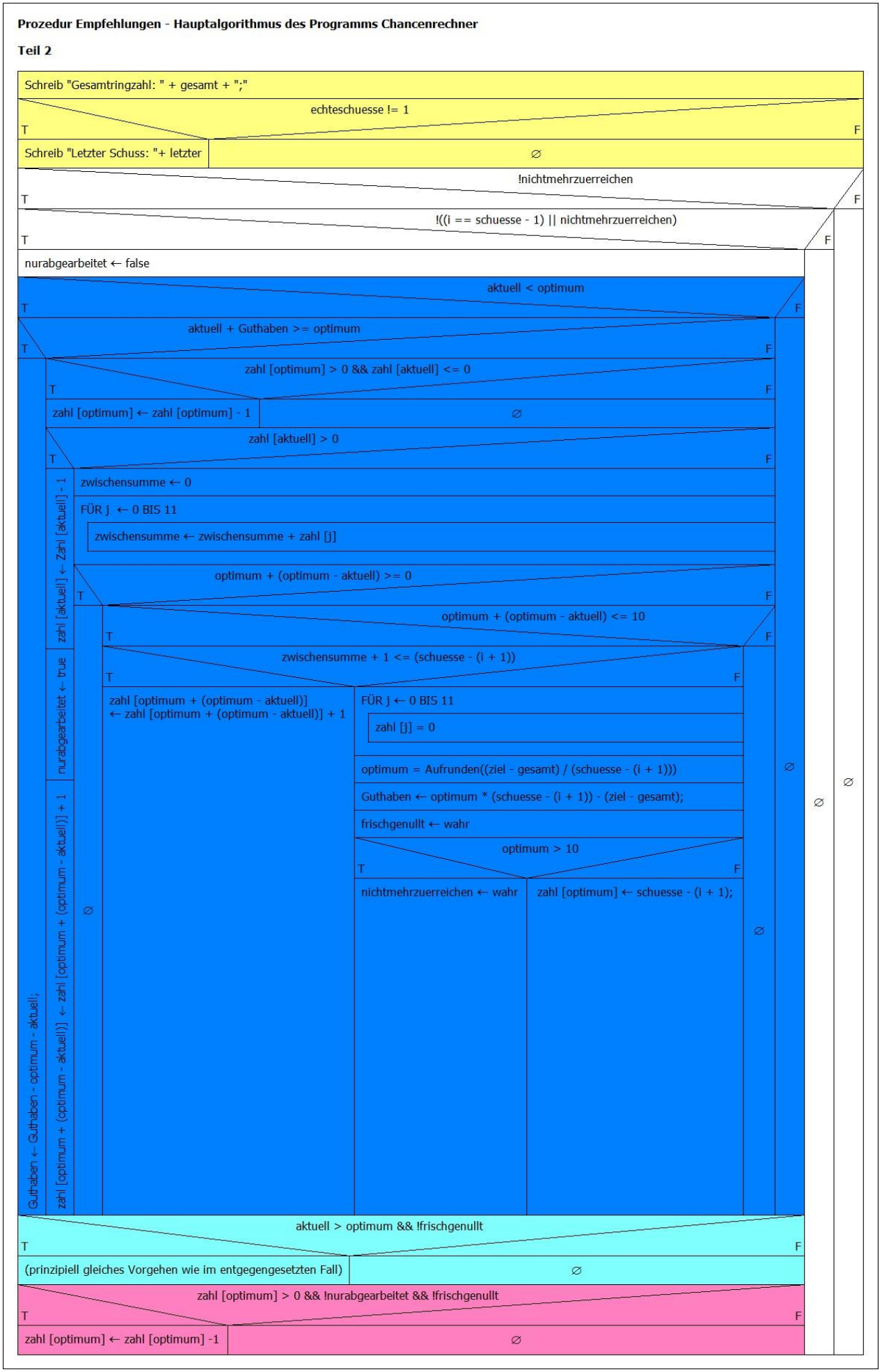
- für n Ringe Ziel bei m Schüssen: Mittelwert ist $\left\lceil \frac{m}{n} \right\rceil$ → immer aufrunden, da man lieber zu viele Ringe schießt als zu wenige
- → Ziel für den Schützen: möglichst immer diesen Wert schießen
- Variable optimum wird bei Eingabe der Schuss- und Ringzahl ermittelt und dem Schützen als Empfehlung gegeben, diesen Wert bei jedem Schuss zu erreichen
- bei jedem Schuss nun 3 Fälle:
  1. Schütze trifft den Optimalwert
  2. Schütze trifft mehr Ringe
  3. Schütze trifft weniger Ringe
- bei Fall 1: Programm behält aktuelle Empfehlung bei
- bei Fällen 2 und 3: Programm muss Empfehlung anpassen
- Idee hierfür: Summe zweier aufeinanderfolgender Schüsse muss 2 * Optimum sein
- → Abweichung vom Optimum beim aktuellen Schuss ermitteln, Empfehlung für nächsten Schuss: Optimum plus oder minus diese Abweichung
- dies kann aber, wenn diese Summe/Differenz die Spanne von 0 bis 10 verlässt, nicht ausführbar sein → Durchschnitt in solchen Fällen neu berechnen
- zusätzlich bei jedem Schuss ausgeben:
  1. den letzten Schuss zur Orientierung
  2. den Minimalwert für diesen Schuss, mit dem man das Ziel noch erreichen kann; berechnet als $verbleibendeRinge - (verbleibendeSchuesse - 1) * 10$

ii. Teil 1: nötige Festlegungen



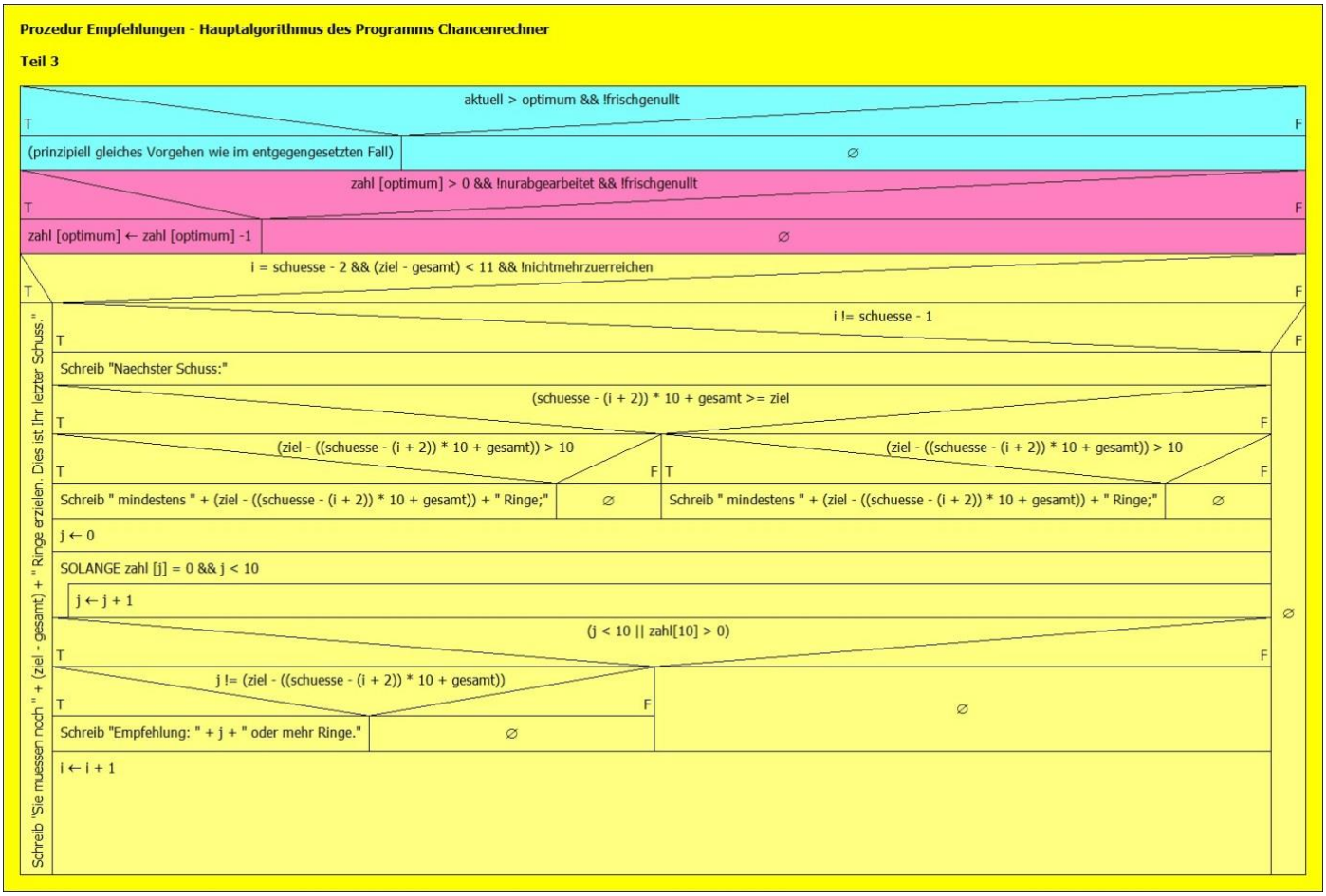
- Bemerkung: Algorithmus wird bei Eingabe jedes Schusses aufgerufen
- Array *zahl* speichert Empfehlungen, dabei ist der jew. Index die Empfehlung und dessen Wert die Anzahl, wie oft das Programm noch empfiehlt, ihn zu schießen
- bei erstem Aufruf: Array initialisieren, sodass es empfiehlt, bei jedem Schuss den Optimalwert zu schießen
- boolean *frischgenullt* → ist wahr, wenn das Optimum gerade neu berechnet wurde
- boolean *falschEingabe* → für ungültige Eingaben, algorithmisch nicht bedeutend, hier aber der Vollständigkeit halber erwähnt
- int *gesamt* → speichert Gesamtringzahl
- bei jedem Schuss: deren Wert um den des aktuellen Schusses erhöhen
- int *letzter* → speichert letzten Schuss; dieser ist der gerade eingegebene, da Empfehlungen für den darauf folgenden Schuss generiert werden
- Array *Ergebnisse* → speichert einzelne Schusswerte
- Algorithmus wird nur ausgeführt, wenn man das Ziel noch nicht erreicht hat

iii. die grundlegende Arbeitsweise des Algorithmus



- Ausgabe der Gesamtringzahl und des letzten Schusses bei jedem Durchlauf
- Berechnung des empfohlenen Schusswertes nur, wenn das Ziel nicht bereits erreicht oder unerreichbar und nicht beim letzten Schuss → sonst Bugs oder unsinnige Empfehlung (z.B. 12 Ringe zu treffen)→ gilt für alle Empfehlungsberechnungen
- wenn dies erfüllt:
    1. Rücksetzen des Booleans, der beschreibt, ob nur eine Empfehlung erfüllt wurde
    2. herausfinden, wie im Bezug zum Optimum geschossen wurde; Annahme: zu wenig geschossen (wohl am häufigsten der Fall)
    3. prüfen, ob die Abweichung mittels des Guthabens ausgeglichen werden kann → falls ja: Guthaben reduzieren, Selektion verlassen
    4. wenn nicht: prüfen, ob die Zahl des Optimalwertes größer 0 ist und die aktuelle Zahl nicht zu schießen war → dann Empfehlungszahl des Optimums erniedrigen, da das Optimum in der folgenden Empfehlung erhalten ist
    5. danach: prüfen, ob aktuelle Zahl zu schießen war → falls ja: Empfehlungszahl der aktuellen Zahl herabsetzen, in boolean speichern, dass Empfehlung befolgt wurde
    6. sonst: Abweichung von Optimum ermitteln und prüfen, ob das Optimum und die Abweichung addiert (der zum Ausgleich nötige Wert) zwischen 0 und 10 liegen; falls ja: Empfehlung, Optimum plus Abweichung zu schießen, erhöhen
    7. sonst: Optimum neu berechnen → Versuch, hier auszugleichen, indem man von er theoretischen Zahl herabgeht und die Empfehlungszahl erhöht, leitet viel zu weit über Ziel hinaus
        - alle Empfehlungen auf 0 setzen
        - Optimum neu berechnen, zwanghaft aufrunden → in keinem Fall zu zuwenig Punkten leiten
        - Guthaben neu berechnen als Differenz der Zahl, die man erhält, wenn man die übrigen Schüsse lang das Optimum schießt, minus des Ziels
        - in boolean speichern, dass gerade ein neuer Optimalwert berechnet wurde
    8. prüfen, ob das Optimum größer ist als 10 → falls ja, speichern, dass man das Ziel nicht mehr erreichen kann; sonst: Empfehlung speichern, für die verbleibenden Schüsse jedes Mal das Optimum zu speichern
- prüfen, ob mehr als das Optimum geschossen wurde → prinzipiell selbes Vorgehen
- Empfehlung des Optimalwerts verringern, wenn dieser noch größer als 0 ist, nicht nur eine Empfehlung (außer das Optimum) geschossen wurde und das Optimum nicht neu berechnet ist → Standardfall; Optimum muss bei jedem Schuss unter diesen Bedingungen verringert werden, tritt ein, wenn Optimum geschossen wurde und sonst, weil das Optimum immer teil der Empfehlung ist

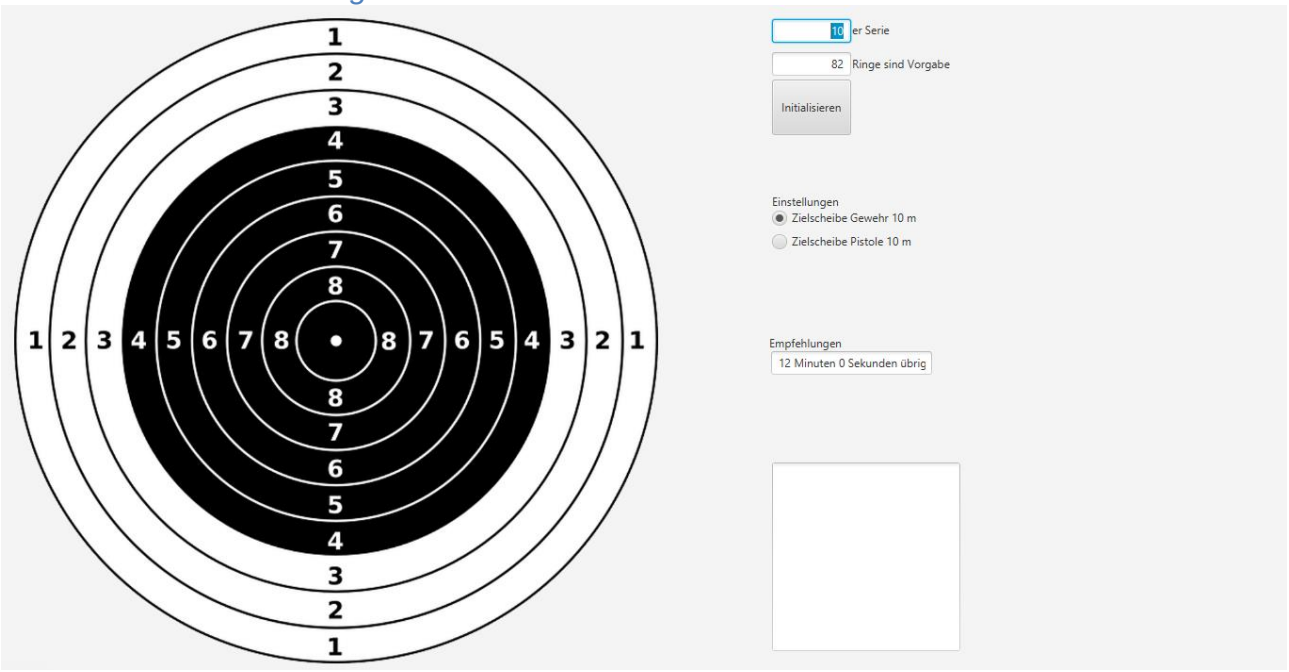
iv. die Generierung der Empfehlung



- Empfehlung generieren
  - wurde gerade der vorletzte Schuss eingegeben und ist das Ziel noch zu erreichen, dann wird als Empfehlung für den letzten Schuss die Zahl der noch zu schießenden Ringe und die Info, dass es sich um den letzten Schuss handelt, ausgegeben
  - sonst: prüfen, ob man nicht im letzten Schuss ist (Zählung beginnt bei 0)
  - Empfehlung für nächsten Schuss ausgeben
    - Information, dass jetzt die Empfehlung kommt
    - prüfen, ob man im nächsten Schuss 0 Punkte erzielen kann (genau dann, wenn man das Ziel noch erreichen kann, wenn man in den Schüssen danach immer 10 Ringe trifft); falls nein: Minimalwert für nächsten Schuss ausgeben; entsteht aus der Differenz aus Ziel und mit den übernächsten Schüssen maximal erzielbaren Punkten
    - kleinste Empfehlung suchen → aus Motivationsgründen diese zuerst ausgeben
    - Empfehlung ausgeben, wenn sie nicht gleichzeitig das Minimum ist
  - Verarbeitungsschusszahl (Zahl der Durchläufe durch HA) erhöhen

c. Umsetzung in JavaFX

i. Umsetzung der grafischen Oberfläche und der Ein- und Ausgabe

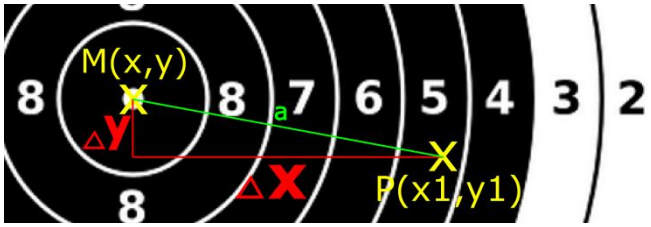


- Aufteilung der Oberfläche in Zielscheibe auf der linken Seite zur Eingabe und Ausgabe und Ein-/Ausgabesegment rechts:
  - Eingabe von Schussserie und Ringziel
  - Einstellung der Zielscheibe (aktuell Gewehr 10m und Pistole 10m)
  - Ausgabe der übrigen Zeit im Textfeld und der einzelnen Schussempfehlungen im TextArea darunter
  - MehrwegButton: Initialisieren → liest Name des Schützen für Protokoll ein, liest Werte für aktuelle Serie (z.B. Schusszahl) ein, prüft diese, fragt evtl. Korrektur ab, speichert diese, gibt Zeitempfehlung und Optimum (Durchschnitt) aus, löscht evtl. vorhandene Werte auf Zielscheibe; Start → startet Stoppuhr und Möglichkeit, Schüsse einzugeben, verhindert Wechseln der Zielscheibe; Stopp → bricht aktuelle Serie ab, stoppt Stoppuhr, ermöglicht wechseln der Zielscheibe, setzt Programm auf Uhrzustand zurück
- läuft das Programm, kann durch Klick auf die Zielscheibe an der Stelle des Treffers dessen Schusswert eingegeben werden; Programm bietet Möglichkeit, bei Fehlern diesen manuell zu korrigieren; Klick startet jeweils Empfehlungsberechnung, nach Eingabe bleibt eine Markierung mit Ort des Schusses und dessen Nummer zurück
- nach Abschluss der Serie wird auf die Zielscheibe Name und Uhrzeit sowie die Auswertung mit Schusszahl, Ziel, erreichten Ringen, Erreichen der Vorgabe und Zeitbedarf
- Zielscheibe mit Schüssen und Bewertung wird am Ende als PNG-Bild auf dem Desktop gespeichert, dazu ein schriftliches Protokoll mit Empfehlungen des Programms; zudem: alle 10 Schuss speichern und leeren der Scheibe, um sie nicht zu sehr zu füllen (Übersichtlichkeit) → Schussergebnisse protokollieren, um später z.B. Entwicklung der Leistungen der Schützen zu untersuchen
- denkbare Erweiterung: speichern von Bild und Text als PDF
- technische Umsetzung der GUI mittels einer auf FXML basierenden JavaFX-GUI
- auf einem Pane liegt ein ImageView (kann ein Bild anzeigen), darauf wird ein Canvas angeheftet (als Zeichenfläche für die Kreise um die eingegebenen Schüsse), an das Pane ist ein MouseListener angeheftet → löst bei Klick Aktion aus

ii. Implementierung weiterer, wichtiger Algorithmen

1. Umsetzen der Eingabe eines Schusswerts

- Bild der Zielscheibe liegt als PNG vor, ist in einem JavaFX ImageView auf der Oberfläche verankert; dieses liegt auf einem Pane, ganz obenauf liegt ein Canvas als Zeichenfläche
- Aufruf dieses Algorithmus bei Klick auf die Zielscheibe durch Nutzer bei laufender Serie
- dabei: Aufruf eines javafx.scene.input.MouseEvent; dieses beinhaltet Koordinaten des Klicks
- Koordinaten der Mitte der Zielscheibe bekannt
- (siehe Bild) Abstand a des eingegebenen Schusses kann ermittelt werden mit $\Delta x = |x1 - x|$ und $\Delta y = |y1 - y|$ durch $a = \sqrt{(x1-x)^2 + (y1-y)^2}$ nach Satz des Pyth.



- Ermittlung des Schusswertes durch Abgleich mit gespeicherten Entfernungen von der Mitte, welche jeweils die Entfernungen der Grenzen der Punktzahlen von der Mitte enthalten → Nachteil des Programms: Bild muss immer in selber Auflösung (800 * 800) dargestellt werden, Zuordnung nach Entfernungen an den Rändern nicht 100% perfekt

2. Umsetzen der Protokollerstellung

- Objekt der Klasse Calender erstellen → enthält Datum und Uhrzeit
- Datum auslesen und in String speichern → Dateiname, davor noch „Protokoll“ setzen
- Statistik mittels Graphics g auf die Zielscheibe schreiben
- Pane flaeche, auf dem Bild und Zeichenfläche liegen, stellt Methode snapshot bereit → liefert Foto der Zielscheibe mit den Markierungen und der Statistik auf ihr
- dieses Bild in WritableImage speichern, dieses mit ImageIO.write auf den Desktop schreiben
- danach Zeichenfläche löschen
- Wiederholung dieser Aktionen alle 10 Schuss → Übersichtlichkeit
- alle Programmein- und Ausgaben wurden im String Protokoll gespeichert
- dessen Inhalt wird in eine Textdatei mit dem selben Dateinamen ebenfalls auf dem Desktop gespeichert

3. Umsetzen der Zeitfunktion

- beim Initialisieren des Programms: Zeit ermitteln (best. Zuordnungen Schusszahl-Zeit, sonst ein Schuss pro Minute)
- Runnable TimerTask enthält eigentliche Aktion, die jede Sekunde auszuführen ist:
    - prüfen, ob noch Zeit ist
        - falls ja: Variable zeit, die verbleibende Zeit enthält, um 1 senken; Restzeit in TextFeld Zeitanzeiger schreiben
        - sonst: Zeitanzeiger den Text „Zeit abgelaufen“ geben
- ScheduledExecutorService exec führt TimerTask jede Sekunde aus
- diese Ausführung ist ScheduledFuture<?> result zugewiesen → Ausführung würde endlos laufen, wenn Programmablauf schon vorbei, result kann Ausführung aber beenden

d. Beschreibung des Arbeitsprozesses

- Erarbeitung in regelmäßigen Treffen mit Trainer und Informatiklehrer
- am Anfang (vor Sommerferien 15): Ideenaustausch, Formulierung einer Spezifikation
- während Sommerferien: Arbeit an Prototypen des Algorithmus
- ursprüngliche Version: ermittelt Empfehlungen nach Abweichung von Optimum (x Ringe zu wenig getroffen → Optimum + x Ringe schießen usw.), wenn zum Ausgleich nötige Zahl kleiner 0 oder größer 10 dann Zahl reduzieren, Anzahl der Schüsse auf diese Zahl erhöhen, Optimum senken, Ausgabe des gesamten Arrays mit den Empfehlungen, Programm als Prototyp auf Konsolenebene in C++ implementiert
- erste Wochen Schuljahr 2015: Verbesserung des Algorithmus (Fehlerbehebung, Einbau der Neuberechnung des Optimums)
- zweites Treffen: Übergabe des Prototypen, Verbesserung diverser Details
- drittes Treffen: Verbesserungen in den Ausgabetermini *altes Protokoll und danach erstelltes Protokoll anhängen* vorgeschlagen, Programm wurde so geändert, dass es immer alle Schüsse durchläuft, statt (wie vorher) abzubrechen, wenn das Ziel nicht mehr erreicht werden kann oder erreicht wurde
- danach: Einbau der Ausgabe eines Minimalwertes, Änderung der Ausgaben (neue Termini und immer nur eine neue Empfehlung)
- Erarbeitung einer GUI zur Demonstration (nur optisch)
- viertes Treffen: Übergabe des Prototypen, diverse Detailverbesserungen, Vorstellung des GUI-Designs
- Implementierung des Hauptalgorithmus in die GUI, Fertigstellung dieser mit allen oben beschriebenen Funktionen
- fünftes Treffen: Detailverbesserungen an der GUI und dem Algorithmus, besonders, dass alle 10 Schuss die Scheibe geleert wird
- Einbau einer Protokollfunktion für die einzelnen Scheiben, bevor diese gelöscht werden
- sechstes Treffen: Detailverbesserungen an der Zeitfunktion und der Oberfläche allgemein
- während der gesamten Erarbeitung: ständige Fehlersuche und -ausbesserung

3. Bewertung des Programms

a. Auswertung der Tests des fertigen Programms

- Programm läuft nach bisherigem Erkenntnisstand stabil, keine Abstürze aufgrund von Fehlern bekannt
- alle Bugs gefixt
- zudem: „Bug-Unempfindlichkeit“ von Java → selbst bei Auftreten kleinerer Fehler kein Programmabsturz, sondern Abbruch der fehlerbehafteten Aktion und Fortsetzung des Programms so weit wie möglich
- Programm zeigt selten Abstürze aufgrund eines sehr schwachen Testrechners → Programm „friert ein“, hängt sich auf, verzögerte Bedienung
- Problem tritt auf stärkeren Rechnern nicht auf, Auftreten auf Testrechner sehr selten und akzeptabel
- teilweise Verzögerungen beim Protokollieren, da die relativ großen Datenmengen der Bildprotokolle erst geschrieben werden müssen; Lösung wäre nur Multithreading (hier kaum sinnvoll) oder eine schnellere Festplatte
- Verzögerungen liegen aber im Bereich der 1-2 Sekunden, sind gerade so spürbar
- → Zusammenfassung: Programm läuft stabil und frei von bekannten Fehlern, selbst bei Auftreten von Bugs verhindern die Exceptiones von Java den Absturz

- *evtl. Trainingserfolge oder Eindrücke der Schützen hier auflisten*

b. Vergleich des fertigen Programms mit seinen Zielen

- Programm gibt dem Schützen sinnvolle und hilfreiche Informationen (lt. Hr. Kunze)
- hilft Schützen, über aktuelle Serie Überblick zu bewahren und die aktuelle Leistung einzuschätzen (letzter Schuss, der allgemeine Schusstrend mit den Markierungen der Schüsse auf der Scheibe…)
- besondere Hilfe bei der Zeiteinteilung durch die Stoppuhrfunktion
- Programm aktuell nutzbar für das Training mit Luftgewehr und Pistole auf 10 m Entfernung → diese sportordnungsgerechten Scheiben aktuell unterstützt, andere leicht einbaubar
- jede mögliche Kombination von Schüssen und Zielringzahl kann simuliert werden
- Zeitvorgaben im Programm basieren auf der deutschen Sportordnung
- Protokolle für Trainer interessant, besonders der Durchschnitt
- Protokolle können archiviert und später erneut zur Auswertung des Trainingserfolges der Schützen herangezogen werden
- Möglichkeit der Erweiterung mit mehr Statistikfunktionen → noch bessere Verwendung der alten Protokolle
- Ein-und Ausgabe relativ genau, Bedienung einfach und Eingabefehlerresistent gelungen
- → Programm hat seine Ziele voll erfüllt und ist eine echte Hilfe für angehende Sportschützen geworden (*oder so etwas in der Art…)*