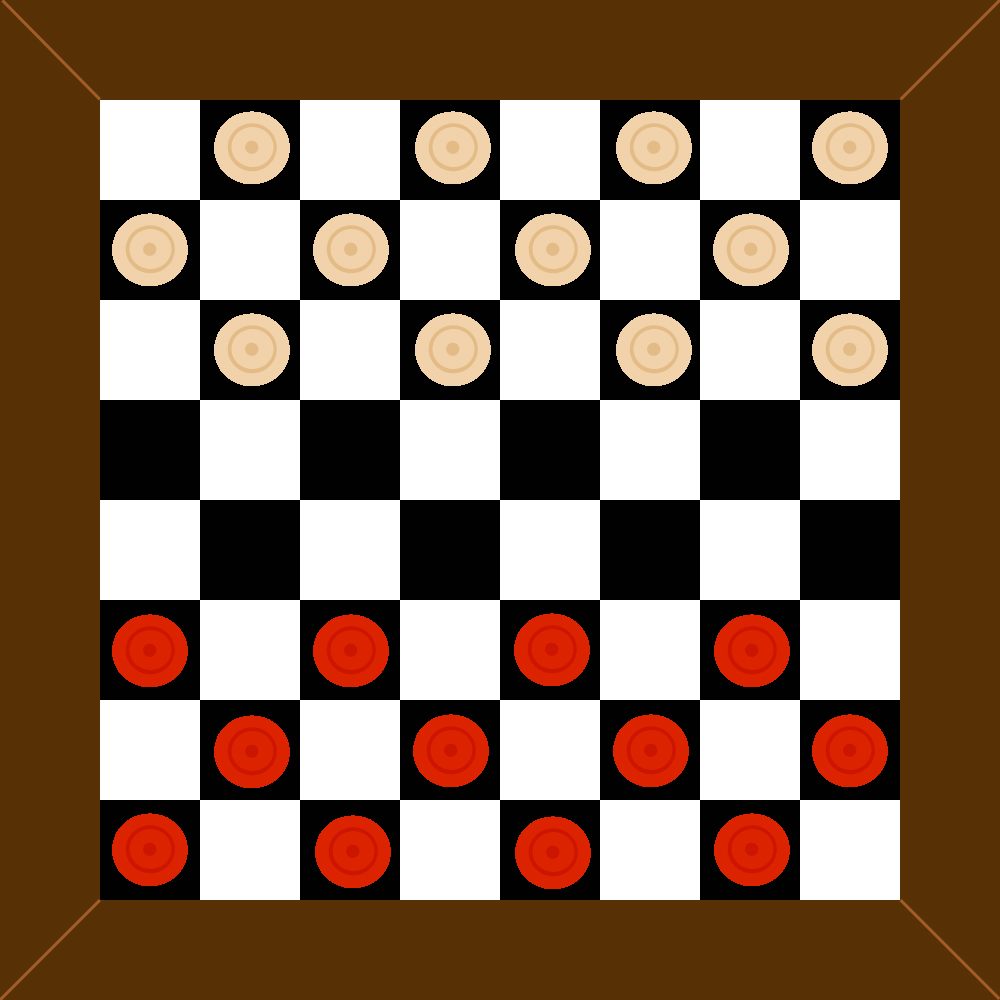
**Bericht Projekt M318**

*Projekt Dokumentation*



Verfasser: Fabio Litscher

Version: 1.0

Erstellungsdatum: 06.06.2014

**Inhalt**

[1. Einleitung 3](#_Toc389834718)

[2. Benutzerschnittstelle 4](#_Toc389834719)

[2.1. Aufgabenangemessenheit 4](#_Toc389834720)

[2.2. Selbstbeschreibungsfähigkeit 4](#_Toc389834721)

[2.3. Steuerbarkeit 4](#_Toc389834722)

[2.4. Erwartungskonformität 4](#_Toc389834723)

[2.5. Fehlertoleranz 4](#_Toc389834724)

[2.6. Individualisierbarkeit 4](#_Toc389834725)

[2.7. Lernförderlichkeit 4](#_Toc389834726)

[3. Fehlermeldungen 5](#_Toc389834727)

[4. Benutzergruppe 6](#_Toc389834728)

[4.1. Beurteilung entsprechend Benutzermodell 6](#_Toc389834729)

[4.2. Nötige Anpassungen / Änderungen 6](#_Toc389834730)

[5. Zustandsdiagramm 7](#_Toc389834731)

[6. Codekonventionen 8](#_Toc389834732)

[6.1. Variablen 8](#_Toc389834733)

[6.2. Funktionen 8](#_Toc389834734)

[6.3. Formular-Steuerelemente 8](#_Toc389834735)

[7. Programm Dokumentation 9](#_Toc389834736)

[7.1. Allgemein 9](#_Toc389834737)

[7.2. Wichtige Funktionen / Bestandteile 9](#_Toc389834738)

[7.2.1. Drei dimensionales Array 9](#_Toc389834739)

[7.2.2. Variante mit Pixel-Koordinaten 9](#_Toc389834740)

[7.2.3. Funktion für die Bestimmung der Koordinaten der einzelnen Felder 10](#_Toc389834741)

[7.2.4. Auf welches Feld habe ich geklickt 11](#_Toc389834742)

[7.2.5. Steine anhand von Array zeichnen 12](#_Toc389834743)

[7.2.6. Wichtigste globale Variablen 12](#_Toc389834744)

[LastColor 12](#_Toc389834745)

[LastPosition 12](#_Toc389834746)

[7.3. Ablauf Regelüberprüfung 13](#_Toc389834747)

[7.3.1. Reihenfolge der zu überprüfenden Regeln 13](#_Toc389834748)

[7.4. Erweiterungsmöglichkeiten 13](#_Toc389834749)

[8. Code Review 14](#_Toc389834750)

[9. Testkonzept 15](#_Toc389834751)

[10. Ergänzungen 16](#_Toc389834752)

[10.1. Besprechungsprotokoll 2. Sitzung 16](#_Toc389834753)

[10.2. Aktualisierte Arbeitsplanung 16](#_Toc389834754)

[10.3. Verwendete Entwicklungsumgebung 16](#_Toc389834755)

[10.4. Datenverwaltung / Datensicherung 17](#_Toc389834756)

[11. Schlusswort 18](#_Toc389834757)

[Abbildungsverzeichnis 19](#_Toc389834758)

# Einleitung

In diesem Projekt, im Rahmen des Modules 318, geht es darum das Brettspiel Dame in der Sprache C# zu programmieren.

Das Ziel ist, dass man mindestens Spieler gegen Spieler spielen kann, ein Zusatz wäre, wenn man Spieler gegen Computer spielen könnte.

Da ich nach drei Wochen bereits alleine in der Gruppe war, da meinem Partner der Lehrvertrag aufgelöst wurde, wurde das Projekt dann von mir alleine realisiert.

Für mich lag die Schwierigkeit darin, dass ich bisher noch nie ein ähnliches Projekt realisiert habe, sowie ich so gut wie keine Programmiererfahrung im objektorientierten Bereich habe.

Das Spiel Dame ist ein klassisches Brettspiel, welches von jeder Altersgruppe gespielt werden kann. Damit die Bedienung möglichst einfach ist, habe ich mich entschieden, das ganze möglichst schlicht darzustellen und mit wenigen, dafür intuitiven Knöpfen die wichtigsten Funktionalitäten zu gewährleisten.

# Benutzerschnittstelle

## Aufgabenangemessenheit

**Gut:**

Unnötige Interaktionen wurden vermieden, indem beispielsweise am Anfang bereits feststeht, dass immer Weiss beginnt.

Standardwerte wie beispielsweise „Spieler 1“ und „Spieler 2“ sind bereits gegeben.

## Selbstbeschreibungsfähigkeit

**Durchschnittlich:**

Hilfen wurden keine grossen eingebaut, allerdings ist das Programm sehr verständlich aufgebaut, da es wenige Knöpfe gibt, bei denen man aber sofort weiss für was sie stehen.

## Steuerbarkeit

**Durchschnittlich:**

Da auf Zusatzfeatures verzichtet wurde, und vor Allem die Grundfunktionalität (Spieler vs. Spieler) gewährleitet wird, gibt es keine grossen Einstellungsmöglichkeiten, welche der Benutzer steuern könnte.

## Erwartungskonformität

**Durchschnittlich:**

Der Grossteil der Knöpfe ist selbsterklärend. Das heisst der Benutzer weiss immer was er macht, und macht nicht ausversehen etwas, was er eigentlich nicht wollte.

## Fehlertoleranz

**Durchschnittlich:**

Es existieren nicht viele Fehlerquellen im Programm, da nicht viele Eingabefelder existieren.

Was allerdings überprüft wird, ist ob der Spieler nach Regeln spielt, man kann also keinen Stein auf ein Feld setzen, wenn man das laut Regelwerk nicht darf.

## Individualisierbarkeit

**Weniger berücksichtigt:**

Der Spieler hat keine grossen Anpassungsmöglichkeiten, die Werte sind vorgegeben.  
Es wurde weniger Wert darauf gelegt, da an erster Stelle die Funktionalität stand.

## Lernförderlichkeit

**Gut:**

Das Programm ist so aufgebaut, dass auch Neuanwender keine Probleme haben das Programm zu bedienen.

# Fehlermeldungen

Fehlermeldungen, wurden bei meinem Programm praktisch keine implementiert, bis auf diejenigen, die in Einsatz treten, wenn man einen unzulässigen Spielzug durchführen möchte. Diese werden jeweils links unten neben dem Spielfeld in der Ereignis-Konsole angezeigt.

Beispielsweise kommt folgende Meldung, wenn der Spieler 1 mit einem seiner weissen Steine, weiter als ein Feld fahren möchte, ohne dass er einen gegnerischen Stein schlagen kann:

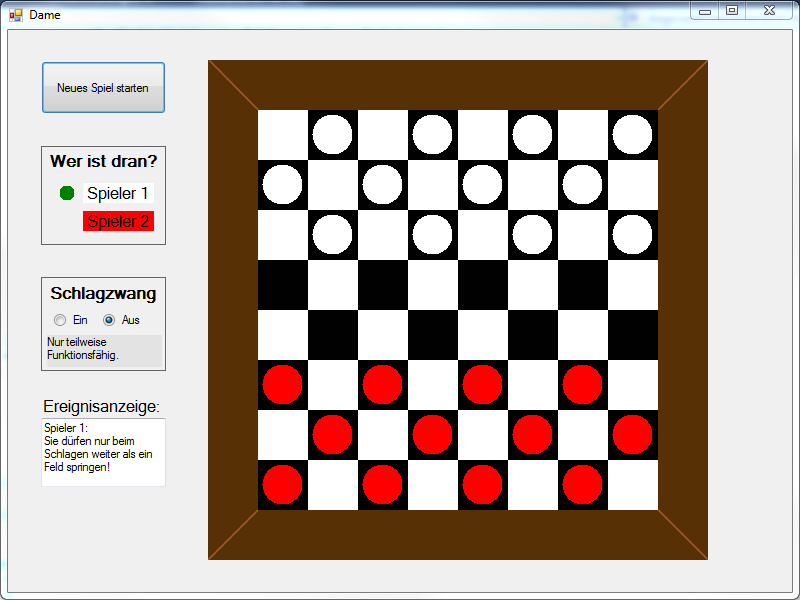


Abbildung 1: Ereignisanzeige

# Benutzergruppe

## Beurteilung entsprechend Benutzermodell

Laut den Definitionen der drei verschiedenen Benutzergruppen, entsprechend dem Benutzermodell (Ergonomie\_BalzertLE16.pdf), würde ich mein Programm folgendermassen beurteilen:

|  |  |  |
| --- | --- | --- |
| **Anfänger** | **Gelegenheitsbenutzer** | **Experten** |
| * Einfache Bedienung durch wenige, dafür klare Funktionen (Knöpfe). * Keine grossen Einstellungsmöglichkeiten. | * Bedienung wie bei anderen gewohnten Programmen. * Keine Schwierigkeiten um zu verstehen, was passiert, wenn man beispielsweise auf einen Knopf klickt. | * Wenn man weitere Funktionen implementieren möchte, ist das eine gute Basis. * Überhaupt keine Probleme bei der Bedienung, findet sich blind zurecht. |
| * Keine Hilfe mit einer ausführlichen Erklärung zum Programm. * Keine interaktiven Hilfestellungen. | * Funktionsumfang evtl. etwas knapp, da kein Modus Spieler vs. Computer möglich ist. | * Geringer Funktionsumfang. * Wenige Einstellungs-möglichkeiten. |

Gemäss dieser Bewertung, würde ich sagen, ist mein Programm hauptsächlich für Anfänger und Gelegenheitsbenutzer entworfen.

## Nötige Anpassungen / Änderungen

Um den Experten-Bereich besser abzudecken, wären folgende Zusätze / Anpassungen nötig:

* Grösserer Funktionsumfang
  + Spieler vs. Computer
  + Speichern von Highscore
  + Spielstand in Datei speichern, Spielstand wieder aus Datei laden
* Mehr Einstellungs- / Personalisierungsmöglichkeiten
  + Möglichkeit zum Anpassen des Farbschemas
  + Beliebiger Name eingeben
    - Macht erst Sinn wenn ein Highscore implementiert wird
  + Skalierbarkeit

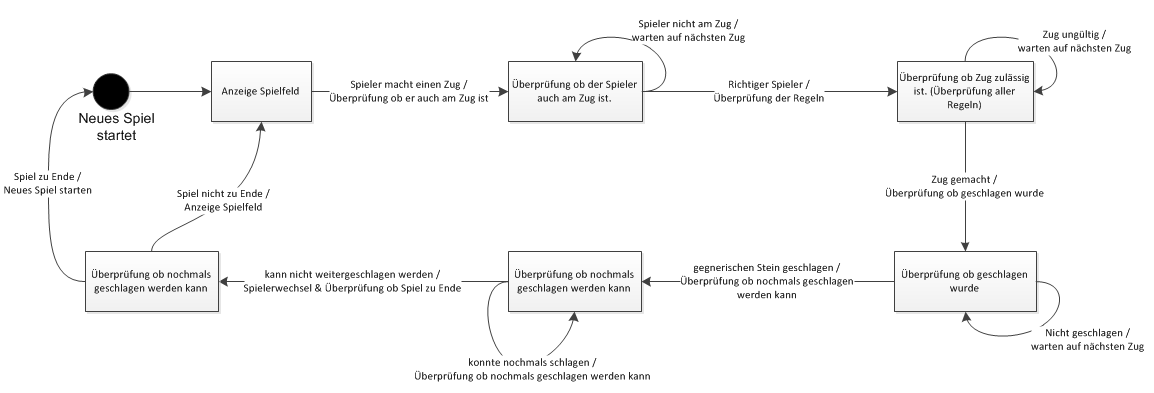
Folgende Sachen wären mit geringem Aufwand möglich zu implementieren:

* Anpassung der Farbe der Spielsteine
  + Mit Hilfe einer kleinen Auswahl aus 4 - 5 fix festgelegten Farben.
* Anpassung des Spieler Namens (anstatt Spieler 1)
  + TextBox anstatt Label machen, dann kann man den Namen bearbeiten.
* Skalierbarkeit
  + Wollten wir anfangs umsetzen, dann aber auf Eis gelegt, da andere Funktionalitäten Vorrang hatten.
* Spielstand in Datei speichern
  + Alle Array Daten in ein Textdokument ausgeben.

# Zustandsdiagramm

Das Zustandsdiagramm meiner Applikation sieht folgendermassen aus.

Das Programm wird beendet, wenn der Benutzer das Fenster schliesst.



# Codekonventionen

## Variablen

Weil bei mir die Verständlichkeit und Orientierung im Programm sehr wichtig war, dass ich mich selber zurechtfinde, habe ich sowohl englische als auch deutsche Bezeichnungen verwendet.

Anfangs wollte ich alles auf englisch machen, habe dann aber gemerkt, dass es für mich einfacher ist, wenn ich das verwende, das mir gerade in den Sinn kommt, sonst muss ich immer überlegen, wie z.B. etwas auf Englisch heisst.

Die Variablen beginnen immer mit einem Kleinbuchstaben.

Es gibt keine Prefixes für globale Variablen oder member Variablen einer Klasse.

Beispiele:

* lastPositionX

Dies können also globale Variablen oder auch member Variablen einer Klasse sein.

* feldX
* fields[]

## Funktionen

Auch bei den Funktionen habe ich sowohl englische als auch deutsche Namen vergeben, auch hier wieder wegen der Verständlichkeit und persönlichen Orientierung in meinem Programm.

Beispiele:

* setFixFields()
* berechneKoordinaten()

## Formular-Steuerelemente

Bei den verschiedenen Steuerelementen habe ich für die verschiedenen Typen die jeweiligen Abkürzungen / Prefixes verwendet.

Der Name ist folgendermassen aufgebaut: *prefix\_name*

Der Name kann sowohl englisch als auch deutsch sein.

Beispiele:

* Textbox 🡪 txt\_console
* Button 🡪 btn\_newRound
* Label 🡪 lbl\_console
* Picturebox 🡪 pic\_spielfeld
* Panel 🡪 pnl\_schlagzwang
* RadioButton 🡪 rdb\_schlagzwangEin

# Programm Dokumentation

## Allgemein

Allgemein ist zu sagen, dass sich alle Funktionen in einer Klasse befinden. Der Grund ist folgender:

Da ich bisher noch nie gross objektorientiert programmiert habe und ich deswegen das ganze Wissen erst aufbauen musste, hat mir Herr Vogel geraten, das Ganze in nur einer Klasse zu programmieren, da so bei Fehlern auch eine Fehlerquelle weniger vorhanden ist und es für mich evtl. etwas verständlicher ist.

Ausserdem ist mein Code eher einfach geschrieben, was aber für meine Verständlichkeit auch von Vorteil war.

## Wichtige Funktionen / Bestandteile

Hier sind nur einige Funktionen und Variablen aufgeführt. Ich habe diese Sachen herausgenommen, die ich persönlich wichtig finde, und die auch meine Grundsteine waren, damit ich überhaupt so weit gekommen bin bei diesem Programm.

### Drei dimensionales Array

Das Herzstück meines Programms ist ein drei dimensionales Array namens fields[].

Die drei Dimensionen:

1. X - Achse
2. Y - Achse
3. Daten

Das heisst das erste Feld links oben wäre das Feld fields[0, 0].

Folgende Daten werden in der dritten Dimension gespeichert:

1. X - Koordinaten 🡪 in Pixel
2. Y - Koordinaten 🡪 in Pixel
3. Gültiges Feld 🡪 0 = zulässig 1 = unzulässig
4. Damenstein 🡪 0 = nein 1 = ja

### Variante mit Pixel-Koordinaten

Da als Spielfeld ein Bild verwendet wird, welches einfach in einer PictureBox ist, und ich irgendwie herausfinden muss, auf welches Feld man klickt, habe ich mich entschlossen, das ganze mit Pixel-Koordinaten zu machen

Der erste Schritt ist herauszufinden, auf welche Koordinaten man klickt, da habe ich anfangs *Mouseposition.X* und *Mouseposition.Y* verwendet. Das Problem bei dieser Variante war, dass die Pixel immer vom Bildschirmrand aus gemessen wurden, das heisst, wenn das Fenster weiter rechts ist, ist die X-Koordinate grösser, als wenn das Fenster ganz links ist.

Zuerst habe ich versucht das auszugleichen mit folgender Formel:

Mouseposition.X - Fensterkoordinate.X - PictureBox.X

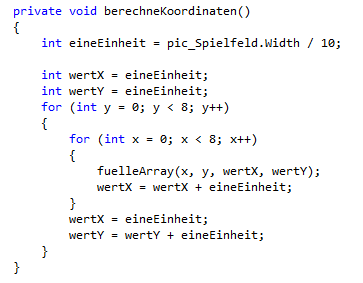
* Hätte theoretisch funktionieren sollen, hat praktisch aber immer um einige  
   Pixel nicht gestimmt.

### Funktion für die Bestimmung der Koordinaten der einzelnen Felder

Wenn das Programm geladen wird, werden in folgender Funktion die Koordinaten der einzelnen Felder errechnet und im Array hinterlegt.

Die Variable **eineEinheit** stellt die Seitenlänge eines einzelnen Feldes dar, diese ist genau 1/10 so gross wie die Seitenlänge der gesamten Picturebox (Da der Rand gleich breit ist wie eine Feld).

Abbildung 2: Funktion für Koordinatenberechnung



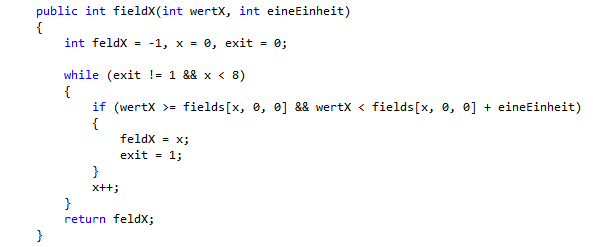
In der for-Schlaufe werden für alle Felder die X- und Y-Koordinaten errechnet, indem immer eine Einheit addiert wird. Die Funktion **fuelleArray()** füllt die Daten dann ins Array ab.

### Auf welches Feld habe ich geklickt

Nun haben wir die Koordinaten des Mausklicks und die Daten der einzelnen Felder sind im Array gespeichert.

Mit folgender Funktion wird berechnet, wo auf der X-Achse man geklickt hat.

Abbildung 3: Funktion für Feldbestimmung



Der Funktion werden der X-Wert des Mausklicks sowie die Länge einer Einheit mitgegeben.

Nun wird immer abgefragt, ob der X-Wert im Bereich eines Feldes liegt, wenn ja geht es aus der Schlaufe raus, wenn nein, wird der Zähler x um eins erhöht, und es wird abgefragt, ob der Klick im Bereich des nächsten Feldes war. Das passiert solange bis ein Feld zutrifft.

Dasselbe passiert natürlich auch noch in Y-Richtung.

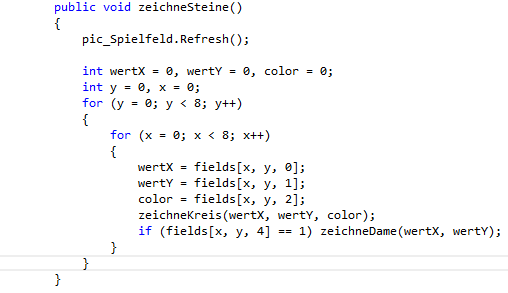
### Steine anhand von Array zeichnen

Nach der Funktion für die Bestimmung des Feldes, war der nächste Schritt, dass ich Steine anhand der Angaben im Array zeichnen konnte.

Diese Funktion sieht folgendermassen aus:

Hier wird immer das ganze Array durchlaufen und die Steine werden entsprechend den Werten im Array gesetzt, indem die Funktion **zeichneKreis()** für das Zeichnen des Spielsteins aufgerufen wird und die Funktion **zeichneDame()** für das Zeichnen des D auf dem Stein, falls es sich um einen Damenstein handelt.

Abbildung 4: Funktion für Zeichnen der Steine



### Wichtigste globale Variablen

LastColor

Mit Hilfe der Variable **lastColor** weiss das Programm immer, was für ein Stein man angehoben hat, da diese Variable immer gesetzt wird, wenn der Spieler der am Zug ist einen Stein anhebt um ihn zu verschieben.

Wenn der Stein dann auf einem leeren Feld (auf welches der Stein laut Regeln gesetzt werden darf) abgesetzt wird, wird diese Variable benötigt, dass dem Feld die richtige Farbe zugeteilt wird.

LastPosition

Die Variablen **lastPositionX** & **lastPositionY** werden immer gesetzt, wenn man auf ein Feld mit einem Stein drauf klickt, damit man weiss, wo der angehobene Stein vorher war.

Diese Variable wird bei den meisten Regelüberprüfungen verwendet.

## Ablauf Regelüberprüfung

### Reihenfolge der zu überprüfenden Regeln

Die Regeln werden in folgender Reihenfolge überprüft:

1. Richtiger Spieler am Zug?
2. Ist das Feld gesperrt?
   * Alle weissen Felder sind gesperrt (In Array hinterlegt).
3. Ist die Fahrtrichtung diagonal?
4. Ist das gewünschte Feld bereits besetzt?
5. Falls es sich um KEINEN Damenstein handelt, ist die Fahrtrichtung vorwärts?
6. Stimmt die diagonale Distanz?
   * Darf nur ein Feld gefahren werden, ausser beim Schlagen wenn ein Stein übersprungen wird.
   * Hier wird auch überprüft, ob man wenn man weiter als ein Feld gefahren ist, einen gegnerischen Stein übersprungen hat. Deshalb gibt es für Damensteine eine separate Funktion, da Damenstein auch rückwärtsfahren und schlagen dürfen.
7. Danach wird der Schlagzwang überprüft.
   * Funktioniert noch nicht zu 100%, deshalb lässt er sich ein- bzw. ausschalten.
8. Wenn man bereits einmal geschlagen hat, wird überprüft, ob man noch weiter schlagen kann.
   * Variable **schongeschlagen** wird auf 1 gesetzt, wenn man schlägt, so kann überprüft werden, ob bereits geschlagen wurde.

Wenn eine Regel greift, sprich verletzt wird, wird der Stein anhand der lastPosition und der lastColor in der entsprechende Farbe auf das Feld gesetzt, von dem er angehoben wurde.

Die Regeln sind ausprogrammiert im Code ersichtlich. Sie wurden hier nicht aufgeführt, da diese Funktionen zum Teil sehr gross sind und es die Dokumentation somit unübersichtlich gestalten würde.

## Erweiterungsmöglichkeiten

Hier sind einige Erweiterungsmöglichkeiten die machbar wären:

* Grösse des Spielfelds der Fenstergrösse anpassen.
  + Wollten wir anfangs implementieren, wurde dann aber runter priorisiert und schlussendlich nicht implementiert. Müsste man mit Pixel arbeiten (Grösse, Koordinaten, ..)
* Farben der Spielstein anpassbar.
  + Wären nur einige Zeilen Code mehr, nicht schwierig zu implementieren, wenn man beispielsweise 5 Farben zur Auswahl stellt.
* „Reallife Modus“
  + Das wäre meiner Idee nach ein Modus, indem alle Regeln ausgeschaltet sind und man spielt wie wenn man auf einem richtigen Spielbrett spielt.

# Code Review

Da ich keinen Projektpartner habe um das Code Review durchzuführen, habe ich meine Mitlernende in meinem Betrieb, welche im 3. Lehrjahr ist gefragt, ob sie das mit mir machen könnte, da es meiner Meinung nach so viel mehr Sinn macht, als wenn ich das alleine machen würde.

Für das Code Review haben wir uns die Funktion schlagenMoeglich() ausgesucht, in der der Schlagzwang überprüft wird. Ein Grund wieso wir diese Funktion gewählt haben, ist, weil sie sehr gross ist und lange if, else Abfragen drin hat, die man evtl. noch vereinfachen könnte.

Genauere Beschreibung:

In dieser Funktion werden alle möglichen Schlagvarianten überprüft, diese setzen sich aus folgenden Sachen zusammen, die alle dazu beitragen, welche Varianten abgefragt werden:

* Handelt es sich um einen Damenstein?
* Um welche Farbe des Steins handelt es sich (weiss oder rot)?
* Ist der Stein zu nahe am linken oder rechten Spielrand, als dass er in Richtung Rand schlagen könnte? 🡪 Dann nur auf die andere Seite überprüfen.
* Ist der Stein zu nahe am unteren oder oberen Spielfeldrand (je nach Farbe)?
* Dann wird immer überprüft, ob ein Feld diagonal zu dem eigenen Stein einer des Gegners ist.
  + Wenn ja, wird abgefragt, ob das nächste Feld auch noch frei ist, damit man den gegnerischen Stein überspringen kann. Wenn das zutrifft, wird dieses Feld in einer Liste gespeichert, in der alle Felder gespeichert sind, auf welche man "schlagen" könnte.
  + Wenn nein, geht die Abfrage weiter zu dem nächsten Stein.

Positives:

* Es sind alle Fälle abgefangen.
* Es ist immer das gleiche Muster, was für Übersichtlichkeit sorgt.

Negatives:

* Die Funktion ist sehr gross.
* Die Abfragen hätten mittels Hilfsfunktionen vereinfacht werden können, da oftmals die gleichen oder ähnliche Abfragen durchgeführt werden.

Erkenntnisse:

Die Erkenntnis ist, dass man die Funktion mit einem etwas grösseren Aufwand bereinigen könnte indem man Hilfsfunktionen macht, die dann die einzelnen Fälle zusammenfassen.

Allerdings müsste man darauf achten, dass die Übersichtlichkeit und das Verständnis weiterhin bestehen würde und nicht unter der Vereinfachung leiden würde.

# Testkonzept

Folgende Sachen wurden getestet:

* Einfacher Zug nach vorne sollte fehlschlagen
* Einfacher Zug auf leeres Feld
* Zug auf Feld ausserhalb des Spielfelds sollte fehlschlagen
* Einfacher Zug rückwärts sollte fehlschlagen
* Einfacher Zug mit Dame rückwärts sollte funktionieren
* Einfacher Zug auf Feld in letzter Linie sollte Stein in einen Damenstein umwandeln
* Einfacher Zug auf besetztes Feld schlägt Fehl
* Angriffszug sollte zwei Felder nach vorne und Feindlichen Spieler töten
* Angriffszug über freundlichen Spieler sollte nicht funktionieren
* Angriffszug über leeres Feld sollte nicht funktionieren
* Angriffszug auf Feld in letzter Zeile ergibt eine Dame
* Nach Angriffszug sollte ein weiterer Angriffszug möglich sein
* Nach Angriffszug sollte kein einfacher Zug mehr möglich sein
* Wenn ein Spieler eine Angriffszug verpasst, sollte dieser gefressen werden können, jedoch nicht wenn der Spieler einen anderen Angriffszug gemacht hat

Die Tests wurden ausgeführt, indem ich die verschiedenen Sachen einfach ausprobiert habe. Zuerst sind die einzelnen Regeln separat ausprobiert worden und dann auch im Zusammenspiel mit den anderen Regeln.

# Ergänzungen

## Besprechungsprotokoll 2. Sitzung

Da ich das Projekt alleine durchgeführt habe, und wir ja eine zweite Sitzung intern in der Gruppe durchführen mussten, habe ich zuerst gedacht, ich beziehe einen meiner Mitlernenden mit ein, damit ich trotzdem eine Sitzung machen kann, allerdings bin ich dann zu dem Schluss gekommen, dass das nicht soviel nützt, da es dann mehr darum ginge, dass ich ihm mein Projekt erkläre.

Wenn Unklarheiten meinerseits bestanden hätten wäre ich auf Herr Vogel zugegangen um eine zweite Sitzung mit ihm durchzuführen.

Da dies aber nicht der fall war, habe ich mich entschlossen keine zweite Sitzung durchzuführen, dementsprechend befindet sich im Ordner *Note 2* auch kein Sitzungsprotokoll der zweiten Sitzung sondern nur das Arbeitsjournal.

## Aktualisierte Arbeitsplanung

Die selben Gedanken wie zur zweiten Sitzung habe ich mir auch zur aktualisierten Arbeitsplanung gemacht. Ich bin dann zu dem Schluss gekommen, dass eine Aktualisierung der Arbeitsplanung vor Allem Sinn macht, wenn man mehrere Personen in einer Gruppe ist, um zu sehen wer was gemacht hat.

Da ich seit ich alleine bin, einfach immer die verschiedenen Punkte nacheinander angegangen bin, empfinde ich es nicht als nötig eine angepasste Version der Arbeitsplanung zu machen.

Was ich der Reihe nach gemacht habe, kann meinem Arbeitsjournal entnommen werden, das ich immer aktuell geführt habe.

## Verwendete Entwicklungsumgebung

Das Projekt habe ich in **Microsoft Visual Studio 2013** durchgeführt, basierend auf der **.NET‑Version 4.5**.

Die Version 4.5 des .NET Frameworks kann hier heruntergeladen werden:

<http://www.microsoft.com/de-ch/download/details.aspx?id=30653>

Microsoft Visual Studio 2013 habe ich mit meinem Gibbix Account heruntergeladen, so konnte ich es gratis beziehen. Sonst ist Microsoft Visual Studio kostenpflichtig und beispielsweise bei Microsoft im Store erhältlich:

<http://www.microsoftstore.com/store/mseea/de_CH/pdp/Visual-Studio-Professional-2013/productID.288483200>

Da das sehr teuer ist, gibt es gratis eine Express Version zum gratis Download:

<http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop>

## Datenverwaltung / Datensicherung

Anfangs hatten wir uns überlegt, wie wir die Daten verwalten, da wir noch zu zweit in der Gruppe waren. Wir haben uns dann für Github entschieden, da wir schon gehört haben, dass das eine gute Variante sei und wir diese einmal ausprobieren wollten.

Wir haben dann das Repository entsprechend angelegt und den Github Client auf unseren lokalen Geräten installiert und es hat alles tip top geklappt. Als ich dann später alleine in der Gruppe war, habe ich mich entschlossen, das mit Github weiterzuführen, da es so sehr gut funktioniert, wenn man an verschiedenen Geräten arbeitet.

So war eine Sicherung auf eine externe Festplatte auch nicht mehr nötig, da man das Repository immer auf die lokale Festplatte geklont hat und ich insgesamt an drei verschiedenen Geräten gearbeitet und somit eine vierfache Sicherheit hatte (3 Geräte & Github).

# Schlusswort

Für mich war das Projekt sehr interessant, da ich bisher noch nie etwas in diese Richtung gemacht habe. Das hat mir auch spass gemacht, weil ich nun am Schluss ein Programm habe, von dem ich sagen kann, dass ich das komplett alleine gemacht und geschafft habe.

Dies erfreut mich, da ich mir anfangs nicht sicher war, wie es rauskommt, als ich mit nur wenigen c# Kenntnissen dieses Projekt gestartet habe und nach zwei oder drei Wochen dann auch schon alleine in der Gruppe war.

Vielleicht war aber auch genau das der ausschlaggebende Punkt, dass ich alleine dann noch möglichst weit kommen konnte.

Mir war am wichtigsten, dass ich mir ein kleines Grundwissen in c# aufbauen konnte und dass ich am Schluss ein zwar noch Verbesserungsfähiges aber fertiges Programm habe, das ich vorzeigen kann.

Die ganze Realisation alleine durchzuführen brachte natürlich auch einen grossen Mehraufwand, da man einfach doppelt soviel machen muss, als wenn man in einer zweier oder sogar dreier Gruppe ist. In meinem Fall hiess das, dass ich nebst der vielen Zeit die ich ins Programmieren investiert habe auch noch das Arbeitsjournal nachzuführen sowie den gesamten Bericht und die Betriebsanleitung zu machen hatte.

Zum Schluss möchte ich noch Herr Vogel danken, dass er mir deswegen in der Bewertung etwas entgegen kommen wird, sonst hätte ich mich wahrscheinlich einer anderen Gruppe anschliessen müssen um den Umfang zu erreichen.

# Abbildungsverzeichnis

[Abbildung 1: Ereignisanzeige 5](#_Toc389834759)

[Abbildung 2: Funktion für Koordinatenberechnung 10](file:///D:\userdata\lfab\My%20Documents\GitHub\M318_Dame\Note%203-1\Bericht.docx#_Toc389834760)

[Abbildung 3: Funktion für Feldbestimmung 11](file:///D:\userdata\lfab\My%20Documents\GitHub\M318_Dame\Note%203-1\Bericht.docx#_Toc389834761)

[Abbildung 4: Funktion für Zeichnen der Steine 12](file:///D:\userdata\lfab\My%20Documents\GitHub\M318_Dame\Note%203-1\Bericht.docx#_Toc389834762)