**Dokumentation Projekt M318**

*Brettspiel Dame (in c# programmiert)*

Verfasser: Fabio Litscher

Version: 1.0

Erstellungsdatum: 03.06.2014

**Inhalt**

[1. Einleitung 3](#_Toc389574215)

[2. Benutzerschnittstelle 4](#_Toc389574216)

[2.1. Aufgabenangemessenheit 4](#_Toc389574217)

[2.2. Selbstbeschreibungsfähigkeit 4](#_Toc389574218)

[2.3. Steuerbarkeit 4](#_Toc389574219)

[2.4. Erwartungskonformität 4](#_Toc389574220)

[2.5. Fehlertoleranz 4](#_Toc389574221)

[2.6. Individualisierbarkeit 4](#_Toc389574222)

[2.7. Lernförderlichkeit 4](#_Toc389574223)

[3. Fehlermeldungen 5](#_Toc389574224)

[4. Benutzergruppe 6](#_Toc389574225)

[4.1. Beurteilung entsprechend Benutzermodell 6](#_Toc389574226)

[4.2. Nötige Anpassungen / Änderungen 6](#_Toc389574227)

[5. Zustandsdiagramm 7](#_Toc389574228)

[6. Codekonventionen 8](#_Toc389574229)

[6.1. Variablen 8](#_Toc389574230)

[6.2. Funktionen 8](#_Toc389574231)

[6.3. Formular-Steuerelemente 8](#_Toc389574232)

[7. Wartung 9](#_Toc389574233)

[8. Programm Dokumentation 10](#_Toc389574234)

[8.1. Allgemein 10](#_Toc389574235)

[8.2. Wichtige Funktionen / Bestandteile 10](#_Toc389574236)

[8.2.1. Drei dimensionales Array 10](#_Toc389574237)

[8.2.2. Feldbestimmung mittels Koordinaten 10](#_Toc389574238)

[8.2.3. Steine anhand von Array zeichnen 10](#_Toc389574239)

[8.2.4. Wichtigste globale Variablen 11](#_Toc389574240)

[LastColor 11](#_Toc389574241)

[LastPosition 11](#_Toc389574242)

[8.3. Ablauf Regelüberprüfung 11](#_Toc389574243)

[8.3.1. Reihenfolge der zu überprüfenden Regeln 11](#_Toc389574244)

[9. Code Review 12](#_Toc389574245)

# 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.

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 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

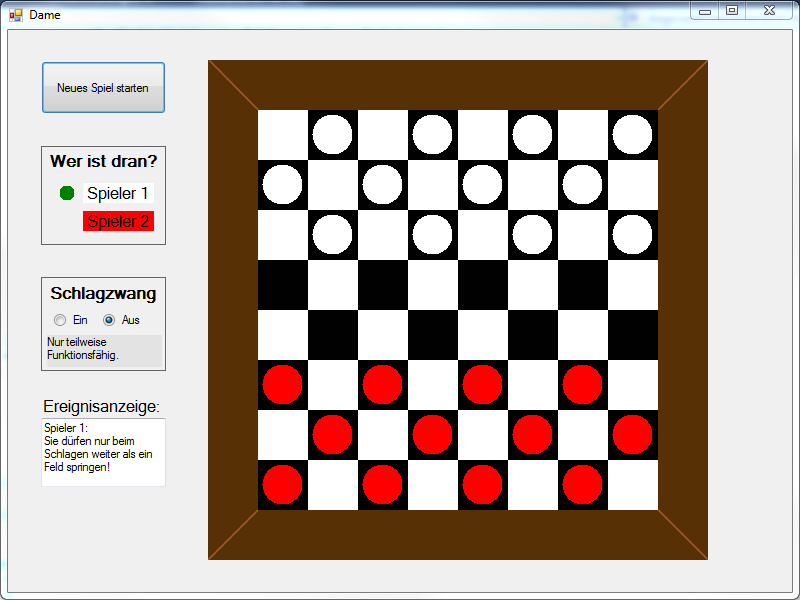
**Gut:**

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:



# 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, alles auskommentiert.
* Spielstand in Datei speichern
  + Alle Array Daten in ein Textdokument ausgeben.

# Zustandsdiagramm

# 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

# Wartung

# Programm Dokumentation

## Allgemein

## Wichtige Funktionen / Bestandteile

### 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

// Bild

### Feldbestimmung mittels 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 rechst 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*

* Theoretisch hätte das funktionieren müssen, was es auch hat, allerdings haben Pixel‑Angaben immer um einige wenige Pixel nicht gestimmt. Da mir das ausgleichen

### Steine anhand von Array zeichnen

### Wichtigste globale Variablen

LastColor

LastPosition

## Ablauf Regelüberprüfung

### Reihenfolge der zu überprüfenden Regeln

# 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.