Pflichtenheft

|  |  |
| --- | --- |
| Projektbezeichnung | Vier gewinnt |
| Projektleiter | Michael Wittner |
| Erstellt am | 03.01.2022 |
| Letzte Änderung am | 20.01.2022 |
| Status | in Bearbeitung |
| Aktuelle Version | 1.1 |
| GitHub Repository | github.com/d-hain/dhain\_sbegic\_viergewinnt |

Änderungsverlauf

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr. | Datum | Version | Geänderte Kapitel | Art der Änderung | Autor | Status |
| 1 | 03.01.2022 | 1.0 | Alle | Erstellung | Wittner Michael | - |
| 2 | 14.01.2022 | 1.1 |  | Korrekturen | 3AHITN | - |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

fg fertiggestellt

iB in Bearbeitung

ab abgebrochen

Pr Prüfung

paus pausiert

Inhalt

[1 Zielbestimmung 3](#_Toc92179561)

[1.1 Musskriterien 3](#_Toc92179562)

[1.2 Wunschkriterien 3](#_Toc92179563)

[2 Allgemeines 4](#_Toc92179564)

[2.1 Ausgangssituation 4](#_Toc92179565)

[2.2 Team 4](#_Toc92179566)

[3 Spielregeln 5](#_Toc92179567)

[4 Funktionale Anforderungen 6](#_Toc92179568)

[4.1 Konsolenanwendung 6](#_Toc92179569)

[4.2 Grafische Anwendung 7](#_Toc92179570)

[4.3 Erweiterung Einzelspielermodus 7](#_Toc92179571)

[5 Nichtfunktionale Anforderungen 8](#_Toc92179572)

[5.1 Programmiersprache / Programmierumgebung 8](#_Toc92179573)

[5.2 Diagramme 8](#_Toc92179574)

[5.3 Programmierrichtlinien / Dokumentation 8](#_Toc92179575)

[5.4 Softwareversionierung 8](#_Toc92179576)

[5.5 Arbeitsbericht 8](#_Toc92179577)

[6 Rahmenbedingungen 9](#_Toc92179578)

[6.1 Zeitplan 9](#_Toc92179579)

[7 Liefer- und Abnahmebedingungen 9](#_Toc92179580)

# Zielbestimmung

Im vorliegenden Pflichtenheft werden alle funktionalen und nichtfunktionalen Anforderungen an das Spiel „Vier gewinnt“ beschrieben.

Es muss eine einfach zu bedienende Anwendung erstellt werden, welche den üblichen Spielregeln des Spieles „Vier gewinnt“ folgt. Die Implementierung hat in den eingeteilten Gruppen zu erfolgen. Um eine einfache Projektorganisation zu gewährleisten, ist ein Softwareverwaltungssystem einzusetzen.

## Musskriterien

* Es müssen zwei Spieler gegeneinander spielen können
* Die Spieler müssen einen Namen eingeben können
* Es muss eine Konsolen- und eine grafische Anwendung erstellt werden
* Die Symbole/Farben der Spielsteine muss wählbar sein
* Eine möglichst komfortable Eingabe eines Spielzuges muss gewährleistet sein
* Eine automatische Überprüfung ob ein Spieler gewonnen hat muss möglich sein

## Wunschkriterien

* Eine UNDO-Funktion muss ermöglicht werden, d.h. ein Zug muss rückgängig gemacht werden können
* Eine Restart-Funktion muss ermöglicht werden
* Ein Einzelspielermodus wird ermöglicht (KI-Modus)

# Allgemeines

## Ausgangssituation

Im letzten Projekt wurde ein Spiel nach dem MVC-Prinzip aufgebaut. Es wurde die Aufteilung in verschieden Klassen vorgegeben.

Innerhalb der Model-Klassen müssen alle für die Spiellogik notwendigen Teile enthalten sein.

Die View-Klassen dienen zur Darstellung der Daten aus den Model Klassen, es darf keine direkte Beziehung zwischen Model und View-Klassen geben. Als Schnittstelle zwischen diesen dient die Controller Klasse. Diese Klasse ist für die Auswertung der Benutzereingaben veranwortlich. Die Eingaben müssen ausgewertet und die entsprechenden Aktionen in den Model Klassen müssen aufgerufen werden.

Nach Aktualisierung der Daten durch das Model müssen diese mit Hilfe der View-Klassen dargestellt werden.

Das vorliegende Projekt muss nach dem gleichen Prinzip aufgebaut werden.

## Team

In der nachfolgenden Tabelle sind die Daten zu den Teammitgliedern einzutragen. Seite 1 ist zu ändern, das Dokument muss in das Git-Repository geladen werden.

|  |  |  |  |
| --- | --- | --- | --- |
| Rolle(n) | Name | Git-Account | E-Mail |
| Projektleitung | Michael Wittner |  |  |
| Entwicklung | David Hain | d-hain | d.hain@gmx.at |
| Entwicklung | Semin Begic | sbegic03 | sbegic@htl-steyr.ac.at |
|  |  |  |  |

# Spielregeln

* Auf einem Feld aus 7 Spalten und 6 Zeilen werfen zwei Spieler abwechselnd einen Stein in eine noch nicht belegte Spalte.
* Der Stein fällt nach unten bis er auf einen anderen Stein oder den Boden trifft.
* Sobald ein Spieler eine Reihe aus vier ununterbrochenen Steinen bildet (waagerecht, senkrecht oder diagonal), gewinnt er das Spiel.
* Kann kein Spieler vier Steine in eine Reihe bringen und sind alle Spalten belegt, so endet das Spiel unentschieden

# Funktionale Anforderungen

## Konsolenanwendung

* Vor Start des Spieles werden die Namen der Spieler eingelesen. Es dürfen nicht zwei Spieler mit dem selben Namen gegeneinander spielen.
* Die Spieler könnnen das Symbol mit welchem Sie spielen auswählen. Die beiden Spieler dürfen nicht die selben Symbole verwenden. Außerdem sollen den Spielern „Standardsymbole“ (‚o‘ und ‚x‘) vorgeschlagen werden.
* Per Zufallsgenerator muss bestimmt werden, welcher Spieler das Spiel beginnt.
* Vor jeder Eingabe der gewünschten Spaltennummer muss das Spielfeld mit dem aktuellen Spielstand dargestellt werden. Der Name des Spielers welcher an der Reihe ist muss ebenfalls ausgegeben werden. Die Eingabe der gewünschten Spaltennummer hat mit Zahlen zu erfolgen. Die Eingabe muss solange wiederholt werden bis eine gültige Spaltennummer (im gültigen Bereich bzw. nicht volle Spalte) angegeben wurde.
* Nach jedem Spielzug muss geprüft werden, ob das Spiel gewonnen wurde. Ist dies der Fall, muss dem Gewinner des Spieles gratuliert werden.  
  Außerdem muss geprüft werden, ob das Spiel unentschieden ausgegangen ist. Dies ist dann der Fall, wenn alle Spalten voll sind, und keiner der Spieler 4 Spielsteine in einer Reihe hat. Eine entsprechende Meldung muss ausgegeben werden
* Nach dem Ende eines Spieles muss nachgefragt werden, ob das Spiel nochmals gestartet werden soll.

**Wunsch-Funktionen**

* UNDO-Funktion: wird anstatt einer Ziffer ein ‚U‘ eingegeben, muss der letzte Spielzug rückgängig gemacht werden. Ein mehrstufiges UNDO, welches es erlaubt mehrere Spielzüge zurückzunehmen, ist nicht vorzusehen.
* Durch die Eingabe von ‚R‘ muss das Spiel neu gestartet werden.

## Grafische Anwendung

* Vor Start des Spieles werden die Informationen zu den beiden Spielern eingelesen:
  + Namen des Spielers
  + Farbe des Spielsteines. Für die Auswahl ist ein systemüblicher Dialog zu verwenden. Standardwerte für die Farben (rot und blau) sind den Spielern vorzuschlagen, welche Sie ggf. ändern können.
* Per Zufallsgenerator muss bestimmt werden, welcher Spieler das Spiel beginnt.
* Nach der Eingabe der Spielerdaten muss das Spielfeld grafisch dargestellt werden. Der Name des Spielers welcher an der Reihe ist, muss in der Anwendung dargestellt werden.
* Die Auswahl der gewünschten Spalte muss mit Hilfe eines Mausklicks ermöglicht werden. Es dürfen nur Mausklicks innerhalb des Spielfeldes und in gültigen Spalten berücksichtigt werden. Wird außerhalb des Spielfeldes geklickt bzw. in eine volle Spalte geklickt, so muss ein Hinweis in der Statuszeile des Programmes ausgegeben werden.  
  Der Hinweis muss durch den nächsten Mausklick wieder gelöscht werden.
* Nach jedem gültigen Spielzug muss geprüft werden, ob das Spiel gewonnen wurde. Ist dies der Fall, muss dem Gewinner des Spieles gratuliert werden.  
  Außerdem muss geprüft werden, ob das Spiel unentschieden ausgegangen ist. Dies ist dann der Fall, wenn alle Spalten voll sind, und keiner der Spieler 4 Spielsteine in einer Reihe hat. Eine entsprechende Meldung muss ausgegeben werden
* Nach dem Ende eines Spieles soll das Spiel durch einen „Restart“ Button neu gestartet werden können.
* Die Anwendung muss ein Menü beinhalten, welches
  + es ermöglicht die Daten zu einem Spieler zu ändern
  + eine Hilfe ausgibt (wie ist das Programm zu verwenden, wie lauten die Spielregeln)

**Wunsch-Funktionen**

* Die UNDO-Funktion muss durch einen eigenen Menüpunkt realisiert werden. Ein mehrstufiges UNDO, welches es erlaubt mehrere Spielzüge zurückzunehmen, ist nicht vorzusehen.
* Ein zusätzlicher Menüpunkt muss den „Restart“ des Spieles ermöglichen

## Erweiterung Einzelspielermodus

Die einfachste Form eines Einzelspielermodus ist es, dass der Computer-Spieler in eine zufällige Spalte wirft. Dieses System ist jedoch nicht wirklich „intelligent“. Eine bessere Möglichkeiten wäre dass der Computer-Spieler die unterschiedlichen Stellungen, die sich nach einem Einwurf ergeben bewertet. Er wirft in die Spalte mit dem besten Zug. Dabei gilt:

* Eine 4er Reihe ist das Ziel – somit die beste Möglichkeit
* Ein 3er im nächsten Zug ist z.B. besser als ein Zug, in dem kein 3er entsteht.
* Ein 2er im nächsten Zug ist besser als ein Zug, indem kein 2er entsteht.

# Nichtfunktionale Anforderungen

## Programmiersprache / Programmierumgebung

Die Anwendung muss in der Programmiersprache Java umgesetzt werden. Für die grafische Anwendung ist JavaFX einzusetzen.

Als JavaFX Version muss die Version verwendet werden, welche am Schulserver unter

//acdc/software/Updates/Microsoft/Windows10/Applications/Java

zu finden ist.

Als Entwicklungsumgebung muss IntelliJ der Firma JetBrains (<https://www.jetbrains.com/de-de/idea/>) eingesetzt werden.

## Diagramme

Für die Erstellung von Diagrammen muss <http://www.draw.io> oder <https://app.genmymodel.com> verwendet werden. Es ist darauf zu achten die Diagramme so in der Versionsverwaltungssoftware zu speichern, dass diese weiter bearbeitet werden können.

Änderungen in Diagrammen müssen dokumentiert (einschließlich Begründung) werden.

## Programmierrichtlinien / Dokumentation

Bei der Programmierung ist auf die Einhaltung der Programmierrichtlinien der HTL Steyr zu achten – siehe

<http://www.htl-steyr.ac.at/intern/wiki/doku.php?id=el-it:fsst:softwareentwicklung:programmierrichtlinien>

Die Inline-Dokumentation hat den Richtlinien zu entsprechen, welche im obigen Dokument beschrieben werden.

Die „Readme.md“ Datei muss alle für das Projekt notwendigen Informationen beinhalten.

## Softwareversionierung

Für die Teamarbeit sowie für die Softwareversionierung ist GitHub zu verweden. Am Ende jedes Unterrichtsblocks müssen Änderungen in das Repository hochgeladen werden. Dabei ist darauf zu achten, dass im main-Branch immer ein lauffähiges Programm existieren muss.

Die Entwicklung der einzelnen Teilbereiche hat in eigenen Branches zu erfolgen. Jedes Teammitglied muss in seinem Branch arbeiten.

Alle Dateien welche für das Projekt notwendig sind müssen in GitHub enthalten sein.

## Arbeitsbericht

Im Repository muss für jedes Teammitglied eine „.md“-Datei für einen Arbeitsbericht erstellt werden. Im Arbeitsbericht muss für jede Unterrichtsstunde von jedem Teammitglied dokumentiert werden, woran gearbeitet wurde, welche Probleme aufgetreten sind, und welche Teile fertig implementiert wurden.

# Rahmenbedingungen

Vor der Implementierung ist ein Klassendiagramm zu erstellen welches alle notwendigen Klassen für die Konsolen- und für die grafische Anwendung beinhaltet.

## Zeitplan

Tag 1 (2h): Besprechung Pflichtenheft

Tag 2 (2h): Entwurf Klassendiagramm, GitHub-Repository

Tag 3,4 (je 2h): Umsetzung/Test Model Klassen

Tag 5 (2h): Umsetzung/Test View Klassen Consolen Anwendung

Tag 6,7 (je 2h): Umsetzung/Test View-Klassen GUI Anwendung

# Abgabebedingungen - Notengebung

Die fertige (Teil-) Software ist am Abgabedatum in GitHub abzulegen.

Im „Main“-Branch ist

* die fertige Konsolenanwendung mit dem Tag „Console“ zu benennen
* die fertige GUI-Anwendung mit dem Tag „Gui“ zu benennen

Nicht lauffähige Projekte werden negativ beurteilt.

Während der Zeit des Projektes werden Aufzeichnungen zu folgenden Bereichen erstellt:

* Einhaltung der zeitlichen Vorgaben
* Arbeiten mit Versionskontrollsoftware
* Kommunikation innerhalb des Teams
* Dokumentation der Fortschritte