GraphColoringGames

Einleitung

Am Ende der Jahrgangsstufe Q2 im Informatikunterricht haben die Schülerinnen und Schüler alle Inhaltsfelder bearbeitet und Gelegenheit zum Erwerb aller notwendigen Kompetenzen erhalten. In einem Leistungskurs kann man dann – auf der Basis z.B. eines Spiels – ein umfangreiches Projekt durchführen, das die Inhaltsfelder „nicht-lineare Datenstrukturen (Graphen)“ und „Netzwerkkommunikation“ bündelt und vielfältige Kompetenzbereiche tangiert.

Insbesondere soll in dem hier vorgestellten Unterrichtsvorhaben ein Netzwerkspiel geplant, modelliert und implementiert werden, bei dem serverseitig die Datenstruktur „Graph“ eine zentrale Rolle spielt. Ebenso wird die Kommunikation zwischen Clients und Server, insbesondere die Entwicklung eines adäquaten Protokolls sowie der damit einhergehenden Server-Algorithmik, eine zentrale Rolle spielen.

Die Spiel-Idee

In der Graphentheorie gibt es diverse Spiele, bei denen zwei (oder auch mehrere) Spieler abwechselnd Knoten bzw. Kanten färben. Unterschiedliche Gewinn- bzw. Verlustregeln garantieren die Vielfalt der Spielmöglichkeiten.

Das folgende Spiel soll hier zunächst als Projektidee benutzt werden:

- Es ist ein Zwei-Personen-Spiel, bei dem die beiden Personen abwechselnd am Zug sind.
- Das Spielfeld ist ein Graph:
  - Er ist ungerichtet.
  - Es gibt keine Mehrfachkanten und keine Schlingen.
  - Die Knoten können mit Farben markiert werden. Dabei stehen – je nach Konfiguration – mehrere Farben zum Knotenfärben zur Verfügung.
- Die beiden Spieler färben abwechselnd einen noch ungefärbten Knoten. Dabei muss die Farbe so gewählt werden, dass kein Nachbarknoten mit dieser Farbe bereits gefärbt wurde.
- Es verliert derjenige Spieler, der unter diesen Bedingungen keinen Zug mehr machen kann.
- Sind alle Knoten korrekt gefärbt, endet das Spiel unentschieden.

Motivierender Hintergrund

Färbungsprobleme auf Graphen sind Gegenstand vielfältiger Forschungen. Neben dem bekannten Vierfarbenproblem kann man aktuell die Arbeit von M. Mirzakhani erwähnen, die sich mit einem der hier benutzten Spielidee verwandten Problem beschäftigt hat. Die Autorin erhielt 1996 als 19-jährige dafür die Fields-Medaille.

Siehe dazu z.B.

- http://www.spektrum.de/magazin/jugendstreich-einer-fields-medaillistin/1420974).
- https://de.wikipedia.org/wiki/Vier-Farben-Satz

Bezug zum Kernlehrplan

Die Schülerinnen und Schüler

- erläutern Operationen dynamischer (linearer und nichtlinearer) Datenstrukturen
- implementieren Operationen dynamischer (linearer oder nichtlinearer) Datenstrukturen
- nutzen die Syntax und Semantik einer Programmiersprache bei der Implementierung und zur Analyse von Programmen
- beurteilen die syntaktische Korrektheit und die Funktionalität von Programmen
- interpretieren Fehlermeldungen und korrigieren den Quellcode
- analysieren und erläutern Protokolle zur Kommunikation in einem Client-Server-Netzwerk
- entwickeln und erweitern Protokolle zur Kommunikation in einem Client-Server-Netzwerk
- wenden didaktisch orientierte Entwicklungsumgebungen zur Demonstration, zum Entwurf, zur Implementierung und zum Test von Informatiksystemen an
- entwickeln mit didaktisch orientierten Entwicklungsumgebungen einfache Benutzungsoberflächen zur Kommunikation mit einem Informatiksystem

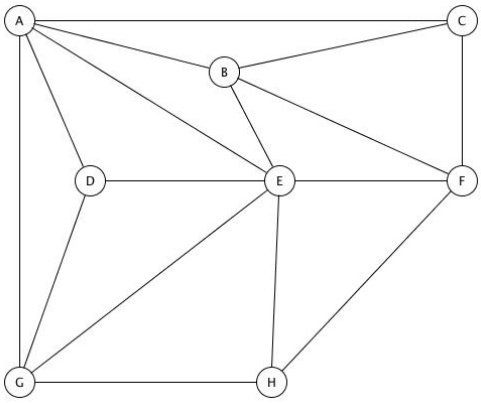
Stufen der Entwicklung

Das hier dargestellte Projekt läuft in mehreren Phasen ab:

1. Das Spiel wird – ohne Computer – auf einem geeigneten Spielplan gespielt. Dabei werden die Spielregeln sorgfältig notiert.
2. In einem Rollenspiel werden der Server und zwei Clients simuliert.
   a. Dabei wird zunächst ein geeignetes Protokoll entworfen, in dem festgelegt wird, welche Nachrichten zwischen Client und Server ausgetauscht werden.
   b. Zu jeder Nachricht, die der Server vom Client empfängt, wird festgelegt, welche Aktion(en) der Server auszuführen hat. Auf diese Weise kann die Implementation der Server-Methode `processMessage` vorbereitet werden.
3. Das Spiel wird ohne Netzwerkanbindung entwickelt:
   a. Es werden notwendige Fachklassen entworfen, andere werden bereitgestellt (z.B. Klassen aus „Grundlegende Materialien und Klassendokumentationen für den Unterricht und für das Zentralabitur“).
   b. Ggf. sind weitere Klassen aus der Java-Bibliothek zu thematisieren. Das könnte zu einer geeigneten Binnen-Differenzierung in dem Kurs beitragen.
   c. Die nicht bereitgestellten Fachklassen werden modelliert, implementiert, dokumentiert und hinreichend getestet.
   Es muss an dieser Stelle möglich sein, das Spiel an einem einzelnen Rechner zu spielen, indem (z.B. mit BlueJ) geeignete Objekte erzeugt und Methoden für diese Objekte aufgerufen werden. Dabei ist die API der Fachklassen auf Vollständigkeit zu überprüfen.
4. Die Klasse `Server` wird per Vererbung spezialisiert und die notwendigen Methoden implementiert.
5. Die Kommunikation mit dem Server wird über einen einfachen Client, z.B. per `telnet`, abgewickelt. Die Nachrichten werden dazu als Zeichenketten, wie im Protokoll festgelegt, an den Server geschickt und die Text-Antworten registriert. Damit wird die Funktionalität des Servers und letztlich das Funktionieren des Spiels getestet.
6. Die Klasse `Client` wird per Vererbung spezialisiert und die notwendigen Methoden implementiert. Ggf. wird eine sinnvolle grafische Benutzeroberfläche implementiert.

1. Das Spiel an einem Beispiel, gespielt auf Papier

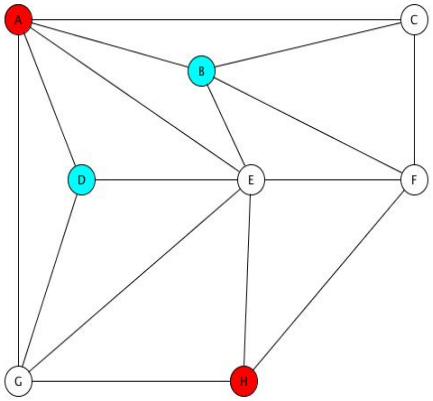
Wir betrachten die oben beschriebene Variante, bei der die beiden Spieler versuchen, die Knoten so zu färben, dass keine benachbarten Knoten identische Farben haben.



Benutzt man zwei Farben (Rot, Blau), so könnte das Spiel auf dem nebenstehenden Graphen folgenden Ablauf haben:

- Spieler 1 färbt den Knoten A mit Rot (1, A, r)
- (2, B, b)
- (1, D, b)
- (2, H, r)

Jetzt kann der erste Spieler keinen weiteren Knoten regelkonform färben. Denn er kann jetzt nur noch einen der (ungefärbten) Knoten C, G, E oder F färben; jedoch blockieren alle Nachbarn dieser vier Knoten beide Farben:



- C hat die gefärbten Nachbarknoten (A,rot) und (B,blau)
- E hat die gefärbten Nachbarknoten (A,rot), (B,blau), (H,rot) und (D,blau)
- G hat die gefärbten Nachbarknoten (A,rot), (H,rot) und (D,blau)
- F hat die gefärbten Nachbarknoten (B,blau) und (H,rot)

Der erste Spieler verliert also das Spiel.

2. Spiel ohne Computer als Rollenspiel

In einer nachfolgenden Phase wird das Spiel, so wie es später im Netzt abläuft, mit verteilten Rollen durchgeführt. Dazu übernimmt eine Person die Rolle des Servers, zwei andere „spielen“ jeweils einen Client. Die Kommunikation erfolgt über den Austausch von schriftlichen Nachrichten. Der Server sowie die Clients visualisieren den Graphen auf Papier.

Ziel dieses Rollenspiels ist es, ein zwischen Server und Client verbindliches Protokoll zu entwickeln. Dabei ist es wichtig, dass die beteiligten Personen (später Objekte vom Typ **`Server`** bzw. **`Client`**) sich strikt an die vereinbarte Syntax des Protokolls halten.

In dieser Version des Spiels gehen wir davon aus, dass – nachdem sich zwei Spieler (Clients) beim Server angemeldet haben – das Spiel automatisch gestartet wird. Der erste Spieler (also derjenige, der sich als erster beim Server gemeldet hat) beginnt. Dann könnte die Kommunikation z.B. folgenden Ablauf haben (siehe Beispiel oben) (C1 bezeichnet den ersten Spieler, C2 den zweiten Spieler, S steht abkürzend für den Server):

Spieler → Server	Server → Spieler
C1: “Färbe A mit Rot”	An C1: “OK” An Alle: “A mit Rot gefärbt” An C2: “Sie sind dran”
C2: “Färbe B mit Blau”	An C2: “OK” An Alle: “B mit Blau gefärbt” An C1: “Sie sind dran”
C1: “Färbe D mit Blau”	An C1: “OK” An Alle: “D mit Blau gefärbt” An C2: “Sie sind dran”
C2: “Färbe H mit Rot”	An C2: “OK” An Alle: “H mit Rot gefärbt” An C1: “Sie sind dran”
C1: “Färbe F mit Rot”	An C1: “OK” An Alle: “F mit Rot gefärbt” An Alle: “Spieler 1 hat gewonnen; Spiel beendet.

Weitere Szenarien müssen hier thematisiert werden, denn jeder Client (also jeder Mitspieler) kann zu jedem Zeitpunkt dem Server eine Nachricht schicken, auf die der Server geeignet reagieren muss. Schickt also ein Client eine Nachricht an den Server, können auch folgende Situationen auftreten:

- Das Spiel ist noch nicht gestartet, da noch nicht zwei Spieler registriert wurden.
- Das Spiel ist bereits beendet.
- Der Spieler, der die Nachricht schickt, ist nicht an der Reihe.
- Der Knoten, der gefärbt werden soll, ist bereits gefärbt.
- Der Knoten, der gefärbt werden soll, kann mit der gewünschten Farbe nicht gefärbt werden, da einer seiner Nachbarn bereits mit dieser Farbe gefärbt ist.
- Der Knoten, der zu färben ist, existiert nicht.

Darüber hinaus kann es sinnvoll sein, dass ein Client vom Server eine Liste aller Knoten (mit deren aktuellen Färbungen) anfordert. Dazu müssen alle Information über alle Knoten in Form einer Zeichenkette (**`String`**) vom Server an den Client geschickt werden.

3. Spiel ohne Netzwerkanbindung; Fachklassen

Nachdem die ersten drei Phasen durchgeführt wurden, ergeben sich Ideen für die Fachklassen, die zu modellieren und zu implementieren sind. Beispielsweise könnten dabei die folgenden Klassen entstehen (die zug. Klassendiagramme sind weiter unten zu finden) :

- **`ColoredVertex`**
  - Eine Klasse, die färbbare Knoten bereitstellt. Objekte vom Typ **`ColoredVertex`** können dann – statt der Standardknoten – in einen Graphen (Objekt vomTyp **`Graph`**) eingefügt und nach entsprechendem „Casting“ wieder ausgelesen werden.

- **`ColoredVertexConstructor`**
  - Diese Klasse stellt Methoden bereit, um aus den Zeilen einer Datei bzw. aus einer Zeichenkette (**`String`**) einen Graphen zu erzeugen, dessen Knoten Objekte vomTyp **`ColoredVertex`** sind. Die Syntax der Eingabe-Dateien bzw. des String-Objektes sind in der Planungsphase genau festzulegen.

- **`EinSpieler`**
  - Notwendige Informationen über einen Mitspieler werden hier verwaltet.

- **`ZweiSpieler`**
  - Die beiden Mitspieler werden in einem Objekt dieser Klasse verwaltet.

- **`GraphSpielfeld`**
  - Ein solches Spielfeld verwaltet den Graphen sowie die beiden Spieler mit Hilfe geeigneter Methoden.

Zu diesem Zeitpunkt muss es möglich sein, das Zwei-Personen-Spiel an einem einzigen Computer, also ohne Netzkommunikation, zu spielen.

4. Spiel im Netz ohne Client

Um das Spiel, also insbesondere den Server zu implementieren, muss ein Protokoll festgelegt werden. Es stellt die Sammlung von syntaktisch festgelegten Nachrichten dar, die der Client an den Server schickt; hier wird auch festgelegt, wie der Server zu reagieren hat und welche Nachrichten er an den Client (oder auch an alle beteiligten Clients) zurücksendet.

Das Protokoll (aus Sicht des Clients) könnte in folgender Tabelle festgelegt werden:

Client an Server	Server an Client(s)	Kommentar
Färbe k mit f	• Falls der Wunsch erfüllbar ist, wird an alle Clients die Nachricht geschickt: *Der Knoten k wurde mit der Farbe f gefärbt* • Fehlermeldung	Der Client möchte den Knoten k mit der Farbe f färben
Knoten	• Eine Liste aller Knoten mit zugeh. Farben wird geschickt	Der Client möchte alle Knoten mit zugeh. Farben erhalten
Graph	• Der Graph wird geschickt	Der Client möchte den gesamten Graphen erhalten

Anschließend entsteht eine Java-Klasse **`Protokoll`**, in der sinnvolle Namen für Nachrichtenformate (Client → Server und Server → Client) als Java-Konstante implementiert sind. Die Klasse **`Protokoll`** könnte dann z.B. Konstanten definieren, die aus dem folgenden Quellcode ersichtlich sind:

```
public class PROTOKOLL {

	// ---------- Nachrichten vom Client an den Server -----------------------------

	// (FF:k:f) Der Knoten k wurde mit der Farbe f gefaerbt
	public static final String CS_FAERBE = "F";

	// (K) Anfrage an den Server: gib mir die Knoteninfos
	public static final String CS_KNOTEN = "K";

	// (G) Anfrage an den Server: gib mir die Grapheninfos
	public static final String CS_GRAPH = "G";

	// (A) Anfrage an den Server: wie viele Farben sind zulaessig?
	public static final String CS_FARBZAHL = "A";

	// -------------- Nachrichten vom Server an den Client --------------------

	// (O:Meldung) mit Trenner werden Meldungen anhehaengt
	public static final String SC_OK = "O";

	// (F:k:farbnr) Der Knoten k wurde mit der Farbe f gefaerbt.
	public static final String SC_GEFAERBT = "F";

	// (K:knoteninfo) Mit Trenner wird der Knotenstring angehaengt
	public static final String SC_KNOTEN = "K";

	// (N:name) Mit Trenner wird der Name des Spielers angehaengt
	public static final String SC_NAME = "N";

	// (G:graphinfo) Mit Trenner wird der Graphstring angehaengt
	public static final String SC_GRAPH = "G";

	// (A:anzahl) Mit Trenner wird die Anzahl der zulaessigen Farben angehaengt
	public static final String SC_FARBZAHL = "A";

	// (E:fehlermeldung) mit Trenner wird die Fehlermeldung anhehaengt
	public static final String SC_ERROR = "E";

	// (U) Das Spiel wurde unentschieden beendet.
	public static final String SC_UNENTSCHIEDEN = "U";

	// (B:ip) Das Spiel ist beendet. Mit Trenner wird die SiegerIP angehaengt.
	public static final String SC_BEENDET = "B";

	// (V) Die Kommunikation mit dem anfragenden Clienten wurde beendet. Ggf.
	// wird noch eine Nachricht per Trenner angehaengt.
	public static final String SC_VERBINDUNG_BEENDET = "V";

	// (S) Das Spiel wird gestartet.
	public static final String SC_START = "S";

	// (Z) Sie sind jetzt am Zug.
	public static final String SC_AKTIV = "Z";

	// (X) Sie sind jetzt am nicht Zug.
	public static final String SC_INAKTIV = "X";

	// ------------------------------------------------------------------------

	// (:) Nachrichten werden immer in der Form TAG TRENNER INFOS verschickt.
	public static final String TRENNER = ":";
}
```

Auf die Nachrichtentypen im oberen Abschnitt muss der Server algorithmisch reagieren. Die Nachrichtentypen im zweiten Teil können von den Clients ausgewertet werden, sofern sie eine Oberfläche haben, in der die Meldungen vom Server benutzerfreundlich aufbereitet werden.

Dann ist der Zeitpunkt gekommen, eine Klasse **`GraphServer`** zu entwerfen und zu implementieren. Dazu muss die vorgegebene Klasse **`Server`** per Vererbung spezialisiert und die drei dort leeren Methoden implementiert werden. Anschließend kann man das Spiel bereits im Netz mit zwei Clients spielen. Dazu kann man einen universellen Client nutzen, der es erlaubt, eine Verbindung zum Server (dessen IP und Port bekannt sein müssen) aufzubauen, mit dem man beliebige Texte zum Server schicken kann, und der die Text-Antworten des Servers unverarbeitet ausgibt. Denkbar ist auch die Kommunikation per `telnet`.

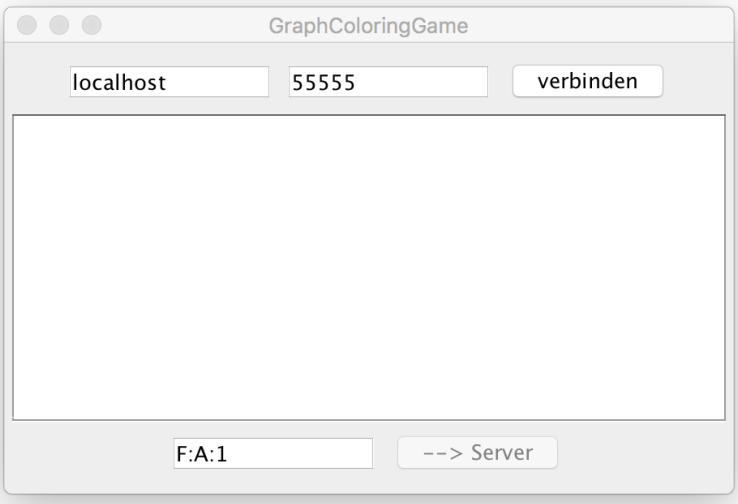
Jetzt zeigt sich, ob das Protokoll hinreichend sinnvoll war, ob die algorithmischen Reaktionen des Servers im Sinne der Spielregeln implementiert wurden und ob die Antworten des Servers an den Client ausreichend sind.

5. Ein einfacher Client

Ein Client, der eine sehr einfache Benutzungsoberfläche zur Verfügung stellt, kann mit den beiden folgenden Klassen realisiert werden:

- **`SimpleGraphClient`**
  - Die Klasse `Client` wird per Vererbung spezialisiert. Die Methode processMessage leitet die vom Server empfangenen Nachrichten unbearbeitet an die grafische Benutzeroberfläche weiter.

- **`SimpleGraphClientFrame`**
  - Diese grafische Benutzeroberfläche hat die Aufgaben, einen Client mit dem Server zu verbinden (Angabe von IP und Port des Servers in den oberen Feldern), dem Server eine Nachricht in Form einer Zeichenkette zu senden (Eingabe in dem unteren Feld) und die vom Server empfangenen Nachrichten (das sind Zeichenketten) im mittleren Feld auszugeben.
    Die zu sendenden Nachrichten sollten der im Protokoll vereinbarten Syntax entsprechen. Jedoch ist der Server (hoffentlich) so modelliert und implementiert, dass auch syntaktisch fehlerhafte Nachrichten nicht zum Server- bzw. Client-Absturz führen.
    Die vom Server empfangenen Nachrichten werden unbearbeitet in einem Textfeld ausgegeben.



6. Ein interpretierender Client

Empfängt der Client vom Server eine Nachricht (also eine Zeichenkette), so kann die Ausgabe benutzerfreundlicher gestaltet werden. Dazu werden auf der graphischen Oberfläche weitere Elemente platziert, die wichtige Informationen für den Benutzer anzeigen:

- Wieviele Farben sind zulässig?
- Welchen Namen hat der aktuelle Client vom Server erhalten?
- Ist der aktuelle Client zur Zeit an der Reihe?
- Welche Fehlermeldung hat der Server geschickt?
- Welcher Knoten wurde (ggf. von anderen Clients) gefärbt? Und mit welcher Farbe?

Die Klasse **`SimpleGraphClient`** wird dazu – bis auf den Klassennamen – unverändert übernommen, so dass sich die Klasse

- **`InterpretingGraphClient`**

ergibt.

Die graphische Benutzeroberfläche ist dann realisiert in der Klasse

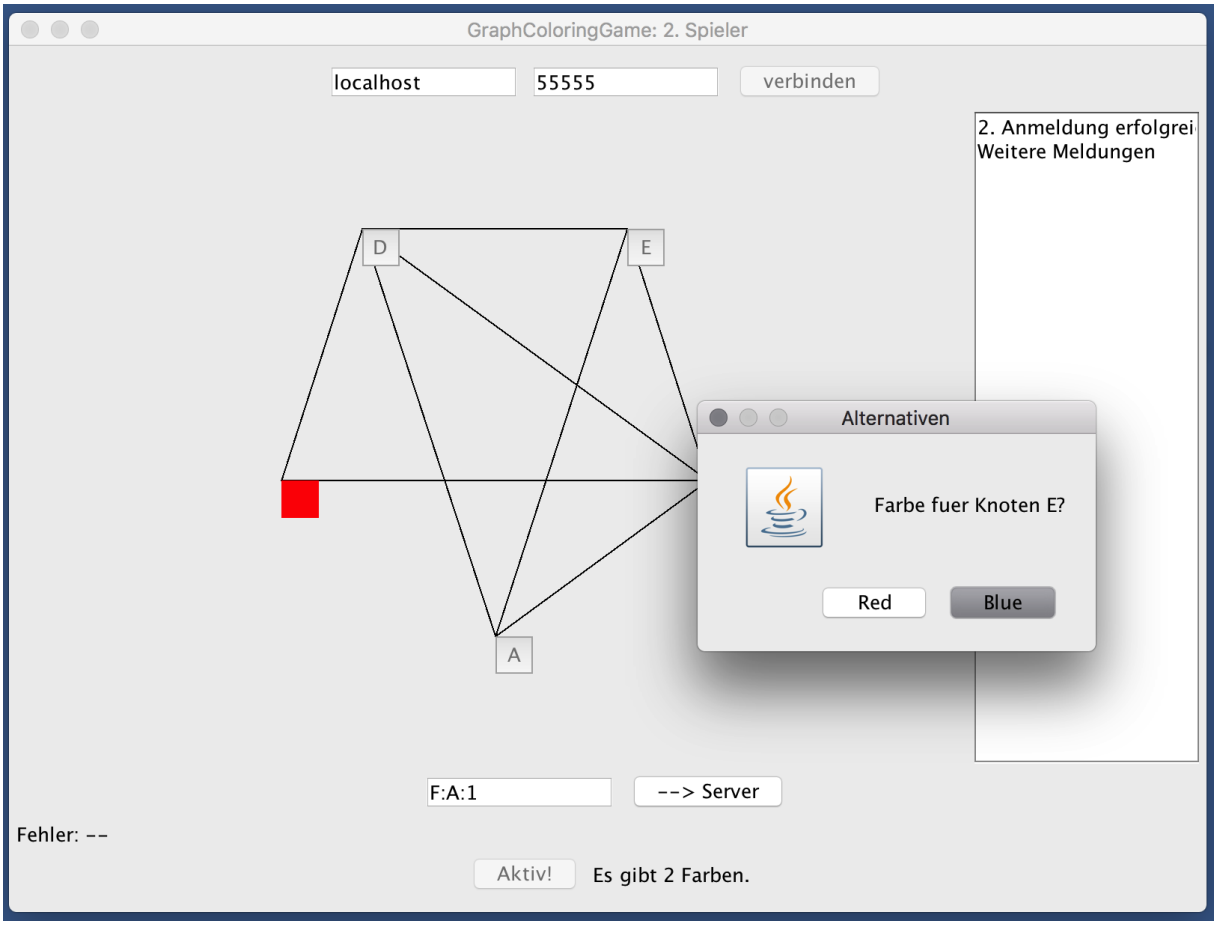
- **`InterpretingGraphClientFrame`**

7. Ein vollständig graphischer Client

In diesem Client wird – zusätzlich zum vorigen Client – der aktuelle Graph dargestellt. Zusätzlich wird die Möglichkeit geschaffen, einen Färbewunsch benutzerfreundlich einzugeben, indem durch Klick auf den Knoten der zu färbende Knoten ausgewählt wird. Anschließend wird über eine Auswahlbox die gewünschte Farbe gesucht. Daneben kann man die Knoten des Graphen (und damit auch die Kanten) selber (nach eigenem Geschmack) anordnen.

Dabei hat der Client keinerlei eigene „Intelligenz“. Also kann der Benutzer z.B. einen bereits gefärbten oder nicht spielregelkonform zu färbenden Knoten zum Färben aussuchen oder einen Knoten mit einer Farbe färben wollen, die nicht erlaubt ist, da ein Nachbarknoten genau diese Farbe hat. Das ist deshalb wichtig, da nur der Server die dazu notwendigen Informationen hat. Natürlich könnte das auch der Client selber überprüfen, da zu Beginn des Spiels jeder Client den kompletten Graphen übermittelt bekommt. Doch die Kontrollinstanz ist immer der Server.

Es ist natürlich denkbar, dass man den Server befragen kann, einen färbbaren Knoten zu benennen, oder für einen Knoten eine gültige Farbe zu verraten. Doch das wäre eigentlich nicht im Sinne der Spielregel.



Damit ergeben sich die Klassen:

- **`ExtendedGraphClient`**
  - identisch mit den bisherigen Clients

- **`ExtendedGraphClientFrame`**
  - Alle Nachrichten des Servers werden geeignet interpretiert (s.o.). Weiterhin wird der Graph (also die Zeichenkette, in dem alle Kanten und Knoten codiert sind), einem weiteren Ausgabeobjekt geschickt, einem Objekt vom Typ:

- **`ExtendedGraphClientPane`**
  - Hier werden alle Knoten und Kanten graphisch dargestellt. Dabei werden die Knoten kreisförmig angeordnet und mit Kanten entsprechend verbunden. Der Benutzer kann alle Knoten beliebig verschieben. Beim Click auf einen Knoten wird nach der gewünschten Farbe gefragt.

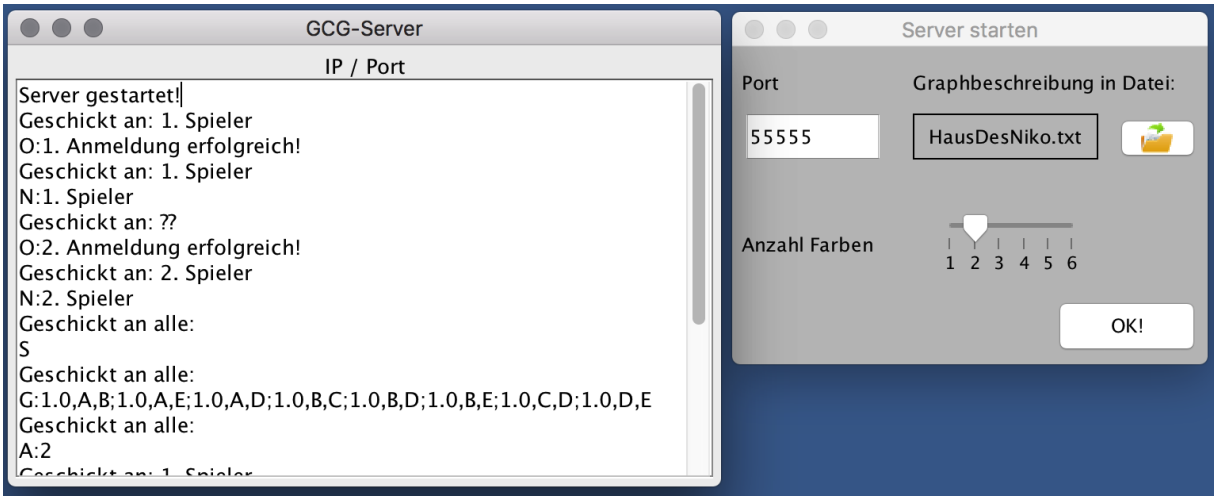
8. Der Server

Die Klasse **`GCG_Server`** ist abgeleitet von der abstrakten Klasse **`Server`**, die in den Materialien aus NRW bereitgestellt wird.

Der Server startet, nachdem folgende Angaben eingegeben wurden:

- Der Port, auf dem der Server zu erreichen ist.
- Die Textdatei, in der der Graph beschrieben ist.
- Die Anzahl der Farben, die benutzt werden dürfen.

Dann startet (zur Kontrolle) ein Fenster, in dem man (auf Serverseite) die Nachrichten, die (aus sicht des Servers) zwischen den Clienten und dem Server ausgetauscht werden, verfolgen kann.



9. Mögliche Erweiterungen und Varianten

Eine sinnvolle Erweiterung besteht darin, einen autonomen Client zu entwerfen. Ein solcher Client sollte, wenn er an der Reihe ist, den Graphen analysieren und einen sinnvollen, regelkonformen Zug machen, ohne dass ein (menschlicher) Spieler über eine Benutzeroberfläche eingreift.

Spielvarianten

Weitere Spiele auf Graphen benutzen andere Spielregeln, die neue Modellierungen und Implementierungen verlangen. Dadurch kann das Projekt – je nach der zur Verfügung stehenden Zeit – erweitert bzw. modifiziert werden. Hier sollen nur einige Varianten vorgestellt werden:

Variante 1: Die beiden Spieler färben Kanten (statt Knoten), wobei Kanten, die einen Knoten gemeinsam haben, unterschiedliche Farben haben müssen. Die Gewinnregel ist analog.

Variante 2: Knoten werden – wie oben – abwechselnd gefärbt. Nachbarknoten dürfen hier jedoch gleiche Farben haben. Es gewinnt derjenige Spieler, der ein Dreieck färbt, dessen drei Knoten gleiche Farben haben.
Ein Dreieck besteht dabei aus drei verschiedenen Knoten k1, k2 und k3, die im Graphen durch einen Kantenzug k1 – k2 – k3 – k1 verbunden sind.

Variante 3: Jeder Spieler hat eine eigene Farbe zum Färben von Knoten und versucht als Erster, ein Dreieck (Viereck, …) mit Knoten der eigenen Farbe zu färben.

Variante 4: Es spielen mehr als zwei Spieler gegeneinander. Die Spielregel ist dabei eine der hier erwähnten.

Ideen für Aufgaben

Die Spielidee kann benutzt werden, um Aufgaben für Klausuren oder für Hausaufgaben zu konstruieren. Beispielsweise könnten Methoden zu entwickeln sein, die folgende Fragen beantworten:

- Ist ein vorgegebener Graph zusammenhängend?
- Können in einem vorgegebenen Graphen alle Knoten regelkonform mit 2 (3, 4, …, n) Farben gefärbt werden?
- Wie groß ist die maximale Anzahl der Knoten, die in einem vorgegeben Graphen regelkonform mit 2 (3, 4, …, n) Farben gefärbt werden können?
- Wie groß ist die minimale Anzahl von Farben, mit denen man einen vorgegebenen Graphen regelkonform färben kann?

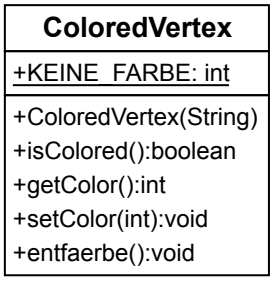
Klassendiagramme

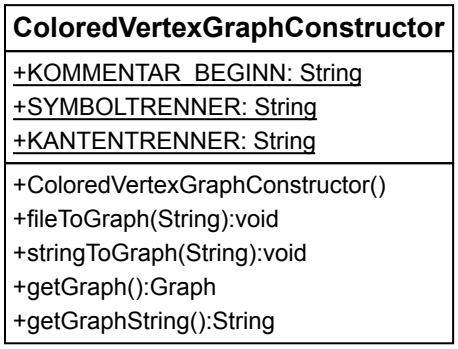
Das gesamte Projekt wurde in der Programmiersprache Java implementiert und u.a. mit der Entwicklungsumgebung BlueJ realisiert.

Um die vielfältigen Klassen zu strukturieren, wurden die Javaklassen in Paketen (mit den angegebenen Java-Klassen) organisiert, wobei geeignete Starterklassen hier nicht aufgeführt sind):

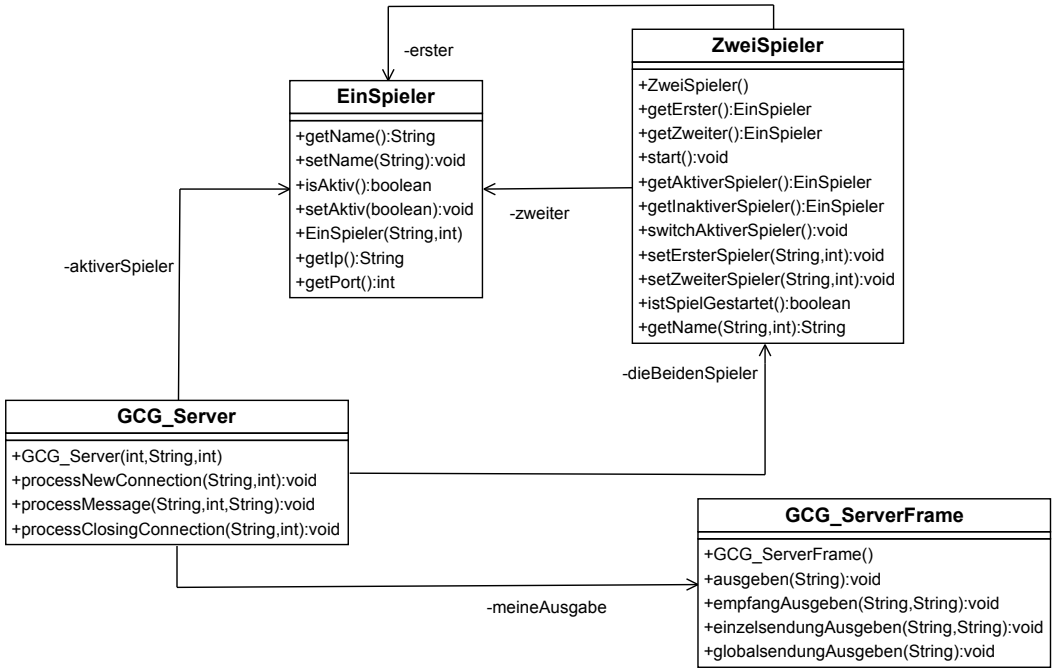
- **`server`**
  - **`EinSpieler`**
  - **`ZweiSpieler`**
  - **`Spielergruppe`**
  - **`GraphSpielfeld`**
  - **`GraphServer`**
  - **`GraphServerFrame`**
- **`protokoll`**
  - **`PROTOKOLL`**
- **`coloredVertexGraph`**
  - **`coloredVertexGraphConstructor`**
  - **`coloredVertex`**
- **`simpleGraphClient`**
  - **`SimpleGraphClient`**
  - **`SimpleGraphFrame`**
- **`interpretingGraphClient`**
  - **`InterpretingGraphClient`**
  - **`InterpretingGraphFrame`**
- **`extendedGraphClient`**
  - **`ExtendedGraphClient`**
  - **`ExtendedGraphFrame`**
  - **`ExtendedGraphPane`**

Abgeleitet aus der Klasse **`Vertex`** und mit Hilfe von Einleseroutinen werden die beiden Klassen **`ColoredVertex`** und **`ColoredVertexGraphConstructor`** entworfen und implementiert.



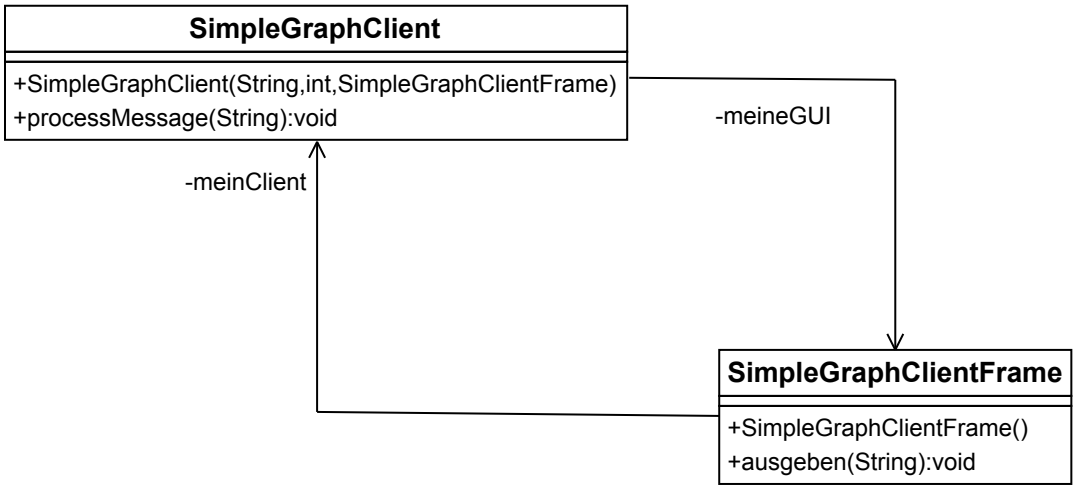


Der **Server** benötigt Fachklassen, um die Spieler zu verwalten. Daneben müssen Spielregeln in den Klassen realisiert werden. Eine GUI sollte mindestens zur Kontrolle des Ablaufs nach Start des Servers existieren.

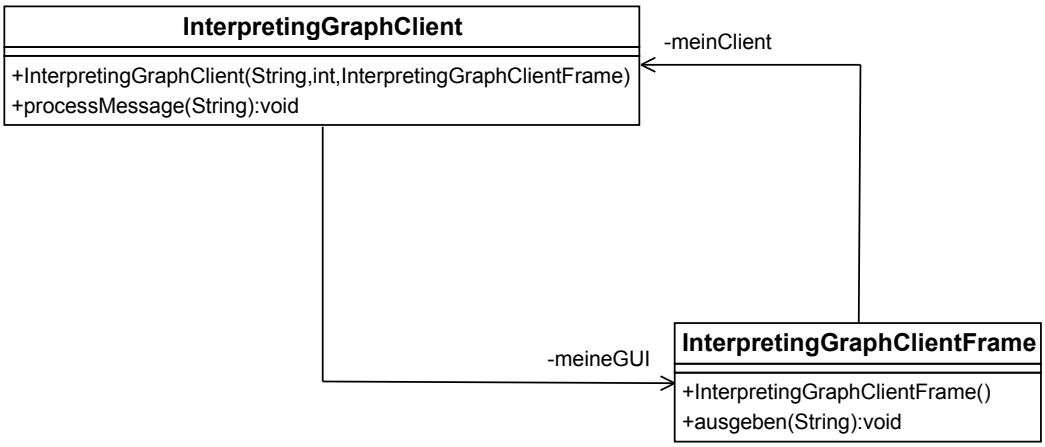


Jeder **Client** besteht im Wesentlichen aus einer grafischen Benutzeroberfläche, mit geeigneten Hilfsklassen. Das kann auf unterschiedliche Arten geschehen:

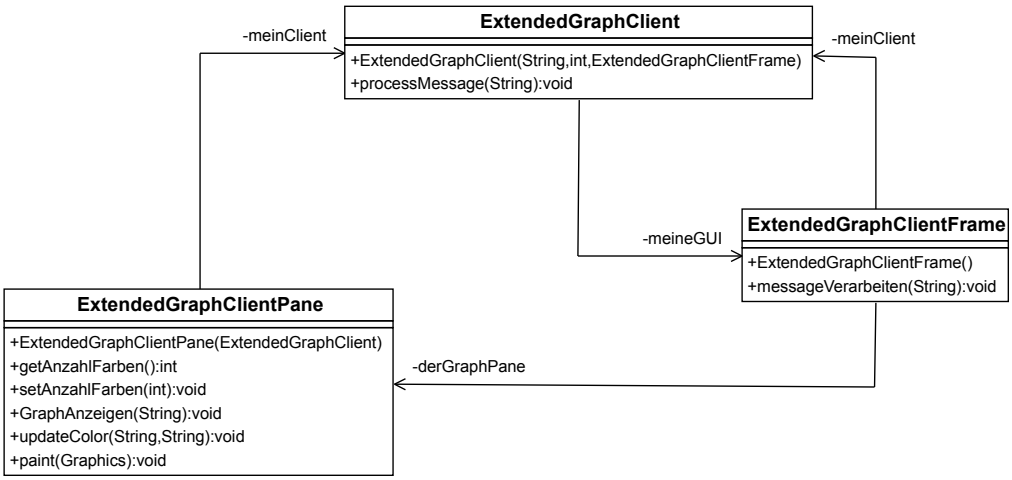
- eine sehr einfache Variante, bei der die Kommunikation ausschließlich textuell abläuft



- in erweiterter Form, bei der im Client-Fenster die Antworten des Servers geeignet interpretiert und benutzerfreundlich dargestellt werden:



- in einer benutzerfreundlichen Version, bei der die Ein- und Ausgaben mit Hilfe des bildlich dargestellten Graphen erfolgen.



Daneben gibt es noch – als Verbindung zwischen den Clients und dem Server – die Klasse **`PROTOKOLL`**, die statische Methoden zum Protokoll bereitstellt: