Programmierpraktikum
Sommer 2023

Department Mathematik / Informatik
Dr. Vera Weil

Meilenstein 2

Abgabetermin: 03.07.2023 um 12:00 Uhr
(Muss um 12:00 Uhr auf dem Server sein!)
Version vom 14. Juni 2023

Bitte suchen Sie für aktuelle Anpassungen nach dem Wort **0612** im Text. Die Buchstaben **a** und **e** symbolisieren Anfang und Ende der jeweiligen Anpassung.

Inhaltsverzeichnis

Bitte suchen Sie für aktuelle Anpassungen nach dem Wort **0612** im Text. Die Buchstaben **a** und **e** symbolisieren Anfang und Ende der jeweiligen Anpassung.

Teil I

Formalitäten, Hinweise und Empfehlungen

1 Formale Vorgaben

1.1 MS2 ist eine Einzelleistung

Erstellen Sie Ihre Lösung selbst und stellen Sie diese nicht zur Verfügung. Andernfalls kann Ihnen das als Täuschungsversuch angekreidet werden.

1.2 Maschinenunabhängigkeit

Alle Ihre Programme müssen maschinenunabhängig laufen. Das heißt insbesondere, dass Sie keine absoluten Pfade verwenden dürfen und dass Sie die Namen der Dateien und Ordner, auf die Sie zugreifen, sowie deren Dateiendungen klein schreiben müssen.
Beachten Sie auch, dass verschiedene Betriebssysteme unterschiedliche Zeichen für Zeilenumbrüche etc. verwenden. Sie müssen alle Ihre `.java`-Dateien in `utf8`-Kodierung speichern; gerade bei Windows ist `utf8` leider nicht standardmäßig in Eclipse eingestellt. Fehler, die auf der falschen Kodierung beruhen, gehen zu Ihren Kosten. Sie **müssen** zudem zum Einlesen von Daten `getResourceAsStream(String name)` oder `getResource(String name)` der Klassen `java.lang.Class` oder `java.lang.ClassLoader` benutzen. Bei der Bezeichnung von Bilddateien muss also alles klein geschrieben sein, auch die Endung.

1.3 Umgang mit dem Repo

In Ihrem Repo finden Sie bereits ansatzweise eine Struktur (in Form von Packages und teilweise vorgegebenen Klassen). Sie dürfen

- ✔ neue Klassen und neue Packages hinzufügen,
- ✔ Klassen aus der Standardbibliothek oder aus Ihrem Projekt in andere Klassen importieren,
- ✔ vorhandene Klassen mit weiteren Methoden, Attributen etc. ergänzen,
- ✔ den Rumpf vorhandener Konstruktoren mit weiterem Code ergänzen.

Sie dürfen **nicht**

- ✖ vorhandene Packages und Klassen umbenennen oder verschieben.
- ✖ vorhandene Attribute umbenennen oder löschen.

- ✖ vorhandene Methodenköpfe ändern oder löschen (insbesondere darf die Signatur einer vorhandenen Methode / eines vorhandenen Konstruktors nicht verändert werden).
- ✖ vorhandenen Code löschen. Das bezieht sich z.B. auf Modifkatoren oder auch auf Importe: Wir importieren z.B. die `Field`-Klasse in die `Testing`-Klasse. Das dürfen Sie folglich nicht ändern. Es gibt eine Ausnahme, wann Sie unseren Code löschen dürfen: Anweisungen, die mit `return` beginnen, dürfen geändert werden.
- ✖ **Klassen in anderen Packages erstellen, die den gleichen Namen haben wie Klassen, die wir schon zur Verfügung stellen.**
- ✖ mehrere Klassen in nur einer Java-Datei erstellen. Das heißt, wann immer Sie eine neue Klasse schreiben, muss diese auch in eine eigene Java-Datei geschrieben werden.

1.4 Java 17, keine externen Importe

Wir kompilieren mit Java 17. Fehler, die aufgrund einer anderen Java-Version entstehen, gehen zu Ihren Lasten. Sie dürfen nur die Java-Standardbibliothek benutzen.

1.5 Modifikatoren

Deklarieren Sie

- alle Getter und Setter sowie Konstruktoren und Klassen als `public`,
- alle von uns verlangten Methoden als `public`,
- alle Attribute (Klassen- und Objektattribute) als `private`, und statten Sie die entsprechende Klasse mit allen entsprechenden Gettern und Settern aus; diese sind nach der Konvention zu benennen (oder generieren Sie sie einfach automatisch mit Eclipse).
- alle abstrakten Methoden in Interfaces ohne Modifikator (sie sind dann automatisch `abstract` und `public`),
- alle Methoden, die in der Hauptklasse stehen, als `public` und `static`,

es seid denn, wir wünschen uns explizit einen anderen Modifikator. Andere Modifikatoren als die von uns vorgeschriebenen können zu hohem Punktverlust führen.

1.6 Ausgaben auf die Konsole

Bei allem, was wir automatisiert testen (Kapitel 10), müssen Konsolenausgaben zwingend vermieden werden. Wir empfehlen dringend, Ihre endgültige Lösung so zu gestalten, dass Sie **keinerlei** Konsolenausgaben mehr verursacht. Haben Sie noch Ausgaben (also `print`-Befehle), die Sie zwecks Selbstkontrolle in Methoden eingebaut haben, so kommentieren Sie diese aus oder löschen Sie diese.

1.7 Name und Matrikelnummer

In **jeder** Ihrer `.java`-Dateien geben Sie ganz oben als Kommentar Ihren Namen und Ihre Matrikelnummer an. Damit bestätigen Sie, dass Sie die Lösung eigenständig verfasst haben.

1.8 Einreichung

Einreichung **nur** mit `git` auf unserem Server. Pushen Sie früh genug. Zum angegebenen Zeitpunkt muss Ihre Lösung bereits auf unserem Server sein, denn zu diesem Zeitpunkt wird die Kopie Ihres Repos gezogen.

1.9 Kompilierbarkeit

Ihr Programm muss kompilieren und in angemessener Zeit terminieren, sonst erhalten Sie dafür 0 Punkte. Sie müssen also insbesondere dafür sorgen, dass **alle** Ihre **Klassen kompilierbar** sind. Wenn eine Klasse nicht kompilierbar ist, so heißt das z.B. im Falle einer automatisierten Überprüfung, dass Sie für **alle** Methoden in der Klasse 0 Punkte erhalten, auch, wenn ein Teil der Methoden funktionieren *würde*, wenn die ganze Klasse kompilierbar *wäre*.

1.10 Umfrage auf Ilias

Vergessen Sie nicht, die Umfrage auf Ilias auszufüllen, die sich darauf bezieht, mit welchem System wir die GUI-Aufgabe von MS2 bewerten sollen (Windows, MacOS), und noch ggf. weitere Hinweise.

1.11 Angemessene Rechenzeit des PCs

Wenn der PC an der Reihe ist, sollte in angemessener Zeit ein Zug gewählt werden. Das dürfte bei langsamen PCs bis zu etwa 4 Sekunden dauern, in der Regel sollte die Wahl aber deutlich schneller erfolgen. Diese Zeiteinschränkung gilt auch für die automatisierte Testung der Strategien. Wenn Sie sich in dieser Hinsicht unsicher sind, kommen Sie gerne in die Fragestunden.

2 Empfehlungen

2.1 Layoutmanager

Wir empfehlen dringend, `LayoutManager` (auch) für das gesamte Frame zu verwenden. Wir empfehlen weiterhin, die Elemente des Fensters in Panel, die ggf. von `JPanel` erben, aufzuteilen.

2.2 Kommentare

Wir empfehlen dringend, Ihre Dateien ordentlich, ggf. sogar im `Javadoc`-Style zu dokumentieren. Beispiele für Javadoc-Kommentare: siehe Kommentare in der API-Spezifikation.

3 Hinweise

3.1 URL für MS2

Angenommen, Ihre S-Mail lautet **mmu@smail.uni-koeln.de**, so ist Ihr S-Mail-Kürzel **mmu**. Die URL für MS2 wäre dann:
ssh://mmu@progserver.informatik.uni-koeln.de:4711/srv/pp/git/pp23/MS2/mmu.git

3.2 Bestehensgrenze

In Aufgabenteil A (Kapitel 9) und B (Kapitel 10) sind jeweils höchstens 530 Punkte zu erreichen. Sie müssen beide Aufgabenteile bestehen, um Meilenstein 2 zu bestehen. Aufgabenteil A gilt als bestanden, wenn Sie mindestens 250 Punkte erhalten haben. Aufgabenteil B gilt als bestanden, wenn Sie mindestens 250 Punkte erhalten haben. Haben Sie Meilenstein 1 und Meilenstein 2 bestanden, so gilt das gesamte Portfolio als bestanden.

3.3 Einsichtnahme zu Meilenstein 2

Das Datum der Einsichtnahme wird noch bekannt gegeben. Sie ist für alle geöffnet, die Meilenstein 1 bestanden haben.

3.4 Widersprüchliches entdeckt?

Im Falle von Widersprüchlichkeiten auf diesem Blatt als auch in Verbindung mit anderen Informationen auf Ilias sind Sie aufgefordert, sich mit dem Team in Verbindung zu setzen.

3.5 Prüfen Ihrer Lösung

Die Prüfung Ihrer Lösung besteht aus zwei Schritten:

- Wir werden die **Start.java** ausführen, die Sie in **start** finden. Daraufhin öffnet sich die graphische Benutzeroberfläche und das Spiel kann gespielt werden. Über diese GUI werden wir die von uns geforderten Funktionalitäten von Hand testen.
- Wir werden die Klasse **Testing** automatisiert testen. Wenn Sie dafür auf Klassen zurückgreifen möchten, die in den anderen Paketen Ihres Repositorys liegen (also mittels **import**), ist das kein Problem und sogar empfehlenswert.

3.6 Erinnerung auffrischen

Schauen Sie ins Orgablatt, um sich gewisse Empfehlungen und Regularien wieder in Erinnerung zu rufen. Wir setzen weiterhin voraus, dass Sie sich regelmäßig auf Ilias über Neuerungen informieren. Auch möchten wir noch mal darauf hinweisen, dass Sie die Fragestunden nutzen

sollten. Sie dürfen des Weiteren nicht davon ausgehen, dass noch kurz vor Abgabedeadline Emails beantwortet werden.

3.7 Gezieltes Auslassen von Anforderungen

Die genannten, zu bearbeitenden Punkte der Aufgaben dürfen Sie nicht als eigene Teilaufgaben sehen, die einzeln bepunktet werden. Vielmehr wird Ihre Lösung - also Ihr Programm - als Ganzes in Hinblick auf die gestellten Anforderungen geprüft und bewertet. Gezieltes Auslassen einzelner Anforderungen kann daher zu einem hohen Punkteverlust führen.

3.8 Fehlende Beschreibungen?

Sollten Sie im Laufe Ihrer Programmentwicklung auf das Problem stoßen, dass etwas nicht beschrieben oder konkretisiert wurde, ist das gleichbedeutend damit, dass Sie sich selbst eine sinnvolle und den Vorgaben nicht widersprechende Lösung überlegen dürfen. Auch beim Design haben Sie im Rahmen der Vorgaben viele Freiheiten. Wir freuen uns über wohlüberlegte Designs. Ein rudimentäres Design, das lediglich dafür sorgt, dass alles gut erkennbar, testbar und funktionstüchtig ist, ist aber auch vollkommen in Ordnung.

- Beispiel: Wenn das Spiel beendet ist, wird ein Hinweis angezeigt (s.u.). Ob das nun bedeutet, dass ein Schriftzug eingeblendet wird, ein Bild, eine Erklärung, oder etwas ganz Verrücktes, ist damit Ihnen überlassen. Es muss für uns als Prüfende nur erkennbar sein, dass Ihr Programm erkannt hat, dass das Spiel als beendet gilt.

Erfahrungsgemäß führen solche „Freiheiten“ aber auch zu einer gewissen Unsicherheit. Natürlich dürfen Sie die Fragestunden nutzen, um diese aus der Welt zu schaffen.

Teil II

Das Spiel - Definitionen, Spielregeln und Spielablauf

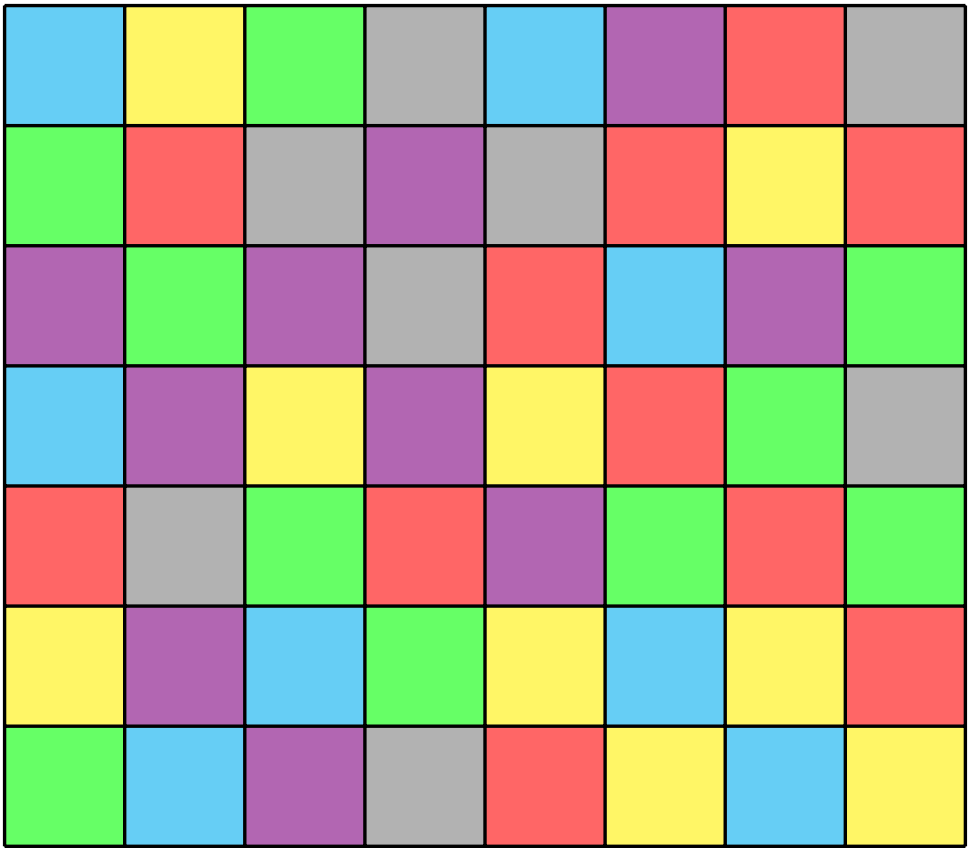

Abbildung 1: Exemplarisches Spielbrett der Größe 7x8

4 Grundlegende Definitionen

4.1 Spieler:in S, S_1, S_2

Mit S bezeichnen wir die Person, die vor dem PC sitzt und das Programm startet und die GUI bedient. Wird ein Spiel gespielt, so unterscheiden wir zwischen S_1 und S_2, die gegeneinander spielen. Die Person S wird zu S_1, sobald ein Spiel gespielt wird. Der PC übernimmt dann die Rolle von S_2. Wenn nicht anders angegeben, beginnt stets S_1.

4.2 Spielbrett

Das Spiel wird auf einem **Spielbrett** gespielt. Dieses Spielbrett (`board`) ist rechteckig und ist in viele kleine, quadratische Felder aufgeteilt. Wird das Spielfenster zu Beginn angezeigt, hat es eine vorgegebene Größe (s.u.). Das Fenster wird verstellbar sein müssen (s.u.). Trotz dieser

Verstellung und der geforderten Responsitivität sollten die Felder dabei stets quadratisch sein. Jedes dieser Felder hat eine eigene Farbe. Siehe z.B. Abb. 1.

4.3 Nachbarn und Wege

Wir nennen zwei Felder A und B zueinander **benachbart**, wenn eine Seite von Feld A an eine Seite von Feld B grenzt. Felder, die sich diagonal berühren, sind also nicht benachbart. Es gibt einen **Weg** zwischen zwei Feldern A und B, wenn A und B identisch sind, oder wenn A und B benachbart sind, oder wenn ein Nachbar von A ein Nachbar von B ist, oder wenn ein Nachbar von A einen Nachbarn hat, der ein Nachbar von B ist, und so weiter. Etwas formaler: Es gibt einen Weg zwischen zwei Feldern A und B, wenn es eine Folge $f_1, ..., f_n$ von Feldern gibt, so dass $f_1 = A$ und $f_n = B$ und f_i ist der Nachbar von f_{i-1} und f_{i+1}, für $i \in \{2, ..., n-1\}$, für $n \geq 1$. In einem Spielbrett (siehe z.B. Abb. 1) gibt es zwischen je zwei Feldern A und B folglich immer einen Weg.

4.4 Zusammenhängend, Fläche und Größe einer Fläche

Wir nennen eine Menge von Feldern **zusammenhängend**, wenn für je zwei Felder in dieser Menge gilt, dass sie über einen Weg miteinander verbunden sind. Eine **Fläche** auf dem Spielbrett ist eine Menge von Feldern des Spielbretts, die zusammenhängend ist. Die **Größe** einer Fläche entspricht der Anzahl der Felder, die in der Fläche enthalten sind.
Zum Beispiel bildet das gesamte Spielbrett (siehe z.B. Abb. 1) eine Fläche, diese Fläche hat die Größe 56. Ein einzelnes Feld ist für sich genommen auch zusammenhängend. ~~Zwei Felder am unteren Rand, die jeweils in einer der beiden Ecken liegen, bilden zum Beispiel keine Fläche, wenn das Spielbrett mindestens 3 Spalten hat.~~ **(0612a)** Genauere Formulierung: Die Menge, die von den folgenden zwei Feldern gebildet wird, bildet zum Beispiel keine Fläche: Die zwei Felder am unteren Rand, die jeweils in einer der beiden Ecken liegen, wenn das Spielbrett mindestens 3 Spalten hat. **(0612e, siehe 11.1)**

4.5 Komponente

Wir nennen eine Fläche im Spielbrett eine **Komponente**, wenn alle Felder in dieser Fläche die gleiche Farbe besitzen. In Abb. 1 bildet jedes Feld eine eigene Komponente. In Abb. 2 bilden alle vier gelben Felder in der rechten, oberen Ecke eine (gelbe) Komponente. Auch die vier violetten Felder in der unteren, linken Ecke bilden eine Komponente.

4.6 Startfelder

Das Spielbrett hat zwei **Startfelder**. Das Feld links unten in der Ecke des Spielbretts ist das Startfeld von S_1 (in Abb. 1 grün gefärbt) und das Feld rechts oben in der Ecke des Spielbretts ist das Startfeld von S_2 (in Abb. 1 grau gefärbt).

4.7 Komponente von S_i und Farbe von S_i

Die **Komponente von** S_i ist die Komponente, die das Startfeld von S_i enthält. Die **Farbe von** S_i ist die Farbe, in der die Komponente von S_i gerade gefärbt ist.

4.8 Zustände des Spielbretts: startklar

Das Spielbrett ist **startklar**, wenn diese drei Bedingungen (gleichzeitig) erfüllt sind:

(1) Für jedes Feld aus dem Spielbrett gilt, dass seine Nachbarn eine andere Farbe haben als das Feld selbst.

(2) Wenn es t viele Farben im Spiel gibt, gibt es auch t viele Farben im Spielbrett.

(3) Das Feld in der Ecke links unten hat nicht die gleiche Farbe wie das Feld in der Ecke rechts oben.

Beispiele:

- Das Spielbrett in Abb. 2 ist nicht startklar.
- Das Spielbrett in Abb. 1 ist startklar, wenn im Spiel auch tatsächlich nur die sechs Farben Blau, Gelb, Grün, Grau, Lila und Rot verwendet werden. Es wäre demnach nicht startklar, wenn es noch eine siebte Farbe - z.B. Schwarz - gäbe, die allerdings in Abb. 1 nicht vorkommt.

4.9 Zustände des Spielbretts: ist Endkonfiguration

Das Spielbrett befindet sich in einer **Endkonfiguration**, wenn alle vorhandenen Felder ~~entweder die Farbe von S_1 oder von S_2 haben.~~ **(0612a)** entweder zur Komponente von S_1 oder zur Komponente von S_2 gehören **(0612e, siehe 11.5)** Das heißt folglich, dass keine der jeweiligen Komponenten mehr vergrößert werden kann.

4.10 Spielzüge: Wahl einer Farbe

In dem Spiel treten S_1 und S_2 gegeneinander an. Sei $i, j \in \{1, 2\}, i \neq j$. Ein Spielzug für S_i besteht darin, dass S_i eine neue Farbe c wählt. Dabei muss S_i sich an folgende Regeln halten:

- S_i darf nicht die Farbe von S_j wählen.
- S_i darf die eigene Farbe nicht noch mal wählen.
- S_i darf unter Beachtung der ersten beiden Punkte aus allen Farben wählen, die im Spiel vorhanden sind (werden unter dem Spielbrett angezeigt).

(0612a) Mit *im Spiel vorhanden* ist die Menge an Farben gemeint, mit denen das Spiel gespielt wird. Es ist unerheblich, welche Farben noch auf dem Spielbrett enthalten sind. Wird also ein Spiel mit 8 Farben gespielt, dann kann S_i jederzeit aus 6 Farben wählen: Es sind nur die Farben ausgenommen, die S_i gerade hat und die S_j gerade hat. Siehe auch 11. **(0612e, siehe 11.2)**

4.11 Spielzug: Wechsel der Farbe von S_i

Sei die Komponente von S_i in der Farbe b gefärbt. Wählt S_i nun die Farbe c, nimmt die bisherige Komponente von S_i nun die Farbe c an. Die neue Komponente von S_i kann nun durch diesen Farbwechsel größer sein als die bisherige Komponente. Das ist genau dann der Fall, wenn zu der bisherigen Komponente Felder benachbart waren, die mit Farbe c gefärbt waren.

4.12 Die Formulierung „genau dann, wenn“

„**Genau dann, wenn** Bedingung B erfüllt ist, gibt eine Methode den Wert `true` zurück.“ bedeutet: Ist Bedingung B erfüllt, wird `true` zurück gegeben. Ist Bedingung B nicht erfüllt, wird `false` zurück gegeben.

4.13 Die Formulierung „die kleinste Zahl“

Jeder Farbe wird eine Zahl zugeordnet. Diese Zahlen sind paarweise verschieden, so dass eine Zahl eindeutig einer Farbe zugeordnet werden kann. Die *kleinste* Farbe bezeichnet also die Farbe, welche durch die kleinste Zahl repräsentiert wird.

5 Ziel und Ablauf des Spiels

In dem Spiel treten zwei Spieler:innen, S_1 und S_2, gegen einander an. Sei $i, j \in \{1, 2\}, i \neq j$. Das Spiel startet, indem ein startklares Spielbrett dargestellt wird. Das Ziel für S_i ist es, am Ende des Spiels mehr Felder zu besitzen als S_j. Das heißt, die Komponente von S_i soll nach Beendigung des Spiels größer sein als die Komponente von S_j. S_i und S_j sind nun abwechselnd an der Reihe. Zur Erinnerung: Standardmäßig startet S_1.

6 Ende des Spiels

Das Spiel endet, wenn eine der beiden Bedingungen erfüllt ist:

(1) Es sind vier Züge hintereinander gespielt worden (2 Züge pro Spieler:in), in denen keine der Komponenten größer geworden ist (dann gilt das Spiel als unentschieden).

(2) Das Spielbrett befindet sich in einer Endkonfiguration.

Ist die Größe der Komponente von S_i größer als die von S_j, so gewinnt S_i. Sind die Komponenten gleich groß, so gilt das Spiel als unentschieden.

7 Beispielhafter Spielverlauf

Betrachten wir nun einen beispielhaften Spielverlauf: Das Spiel startet z.B. mit dem Spielbrett aus Abb. 1. S_1 und S_2 haben jeweils eine Komponente der Größe 1. S_1 beginnt und wählt die Farbe Lila. Die Komponenten von S_1 wird damit nicht größer. S_2 wählt die Farbe Rot. Daraufhin vergrößert sich die Komponente von S_2 auf 3. S_1 wählt anschließend die Farbe Blau und hat damit eine Komponente der Größe zwei. S_2 wählt nun Gelb (neue Komponentengröße: 4). S_1 wählt nun Lila (neue Komponentengröße: 4; S_1 durfte Gelb nicht wählen, da die Farbe von S_2 gerade Gelb ist). Der Status Quo nach diesen fünf Zügen ist nun in Abb. 2 dargestellt.

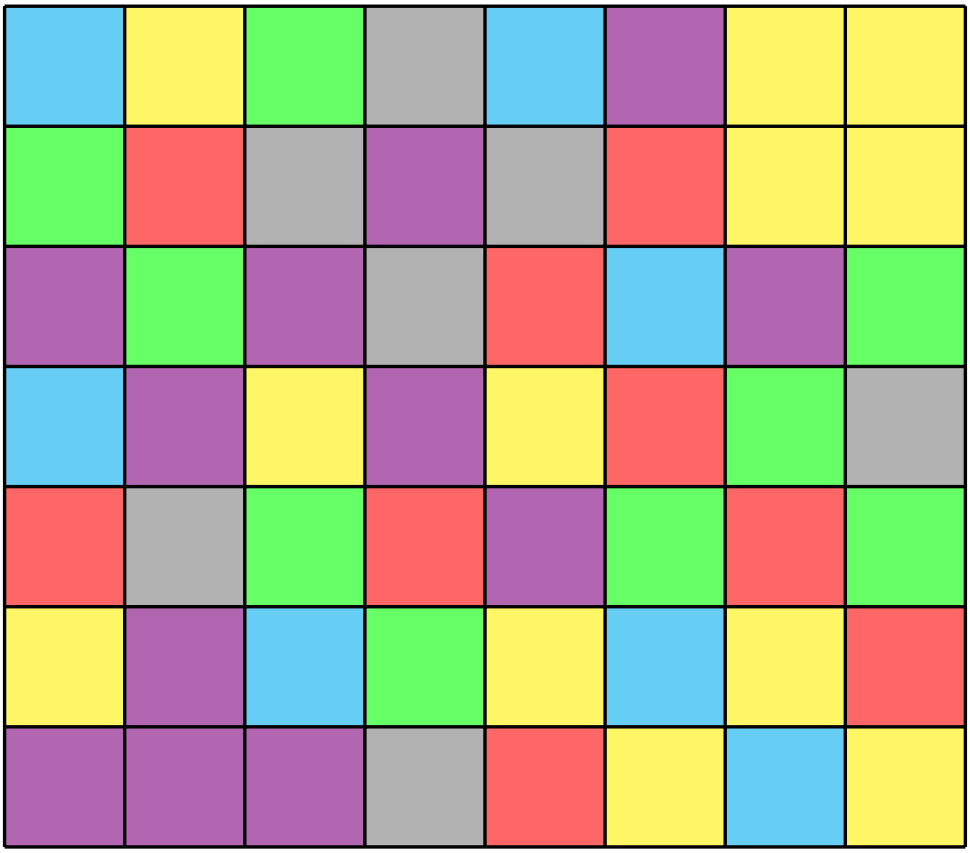

Abbildung 2: Beispielhafter Spielverlauf nach fünf Zügen

8 Mögliche Strategien

Es gibt verschiedene Strategien, die bei diesem Spiel verfolgt werden können. Ein paar dieser Strategien beschreiben wir hier. Sie werden diese **(0612a)** aus Sicht von S_2 **(0612e, siehe 11.4)** implementieren müssen. **(0612a)** Sie dürfen davon ausgehen, dass wir die Strategien nur mit solchen Spielbrettern testen werden, die aus einem Spiel heraus entstanden sein könnten. **(0612e, siehe 11.9)**

8.1 Strategie01: Stagnation

Folgt S_i dieser Strategie, so soll die Komponente von S_i im Wesentlichen nicht größer werden. Es wird also die Farbe gewählt, welche die geringste Vergrößerung (also auch keine Vergrößerung, insofern möglich) zur Folge hat. Stehen mehrere Farben dafür zur Verfügung, wird die kleinste Farbe gewählt.

- Beispiel: Wir nehmen hier an, dass das Spiel, das gerade gespielt wird, nur mit den Farben gespielt wird, die auch gerade im Spielbrett aus Abb. 1 angezeigt werden. Bei dieser Strategie muss **(0612a)** S_2 **(0612e, siehe 11.10)** in Abb. 1 eine Farbe wählen, die nicht identisch ist zu Grau, Grün oder Rot. Ist zum Beispiel Lila mit 2, Hellblau mit 3 und Gelb mit 5 repräsentiert, wäre Lila als die kleinste Farbe die richtige Wahl.

8.2 Strategie02: Greedy

Verfolgt S_i diese Strategie, so wählt S_i stets die Farbe, welche die Komponente von S_i aktuell am meisten vergrößert. Stehen mehrere Farben zur Auswahl, wird die kleinste Farbe gewählt.

- Beispiel: Bei dieser Strategie muss S_1 in Abb. 1 die kleinste Farbe aus Blau oder Gelb wählen.

8.3 Strategie03: Blocking

Verfolgt S_i diese Strategie, so wählt S_i stets die Farbe, welche die **(0612a)** ~~Farbfläche~~ Komponente **(0612e, siehe 11.6)** von S_j am meisten vergrößern würde, wenn S_j am Zug wäre. Stehen mehrere Farben zur Auswahl, so wird die kleinste Farbe gewählt.

- Beispiel: Bei dieser Strategie muss S_1 in Abb. 1 Rot wählen, so dass S_2 nun nicht mehr Rot wählen kann.

Teil III

Die Implementierung des Spiels

Implementieren Sie das beschriebene Spiel. Es soll nun möglich sein, es über eine graphische Benutzeroberfläche zu bedienen und das Spiel graphisch nachzuvollziehen. Wir empfehlen dringend, sich vorher eine Struktur zu überlegen, wie Sie das Ganze aufbauen. Schauen Sie auch in die Programmierstarthilfen (Stichwort: Daten - Logik - GUI). Ein eigenes Packages für die `gui` wäre beispielsweise eine Möglichkeit, um Ihrem Programm ein wenig Struktur zu geben. Ein wenig Struktur haben wir Ihnen auch schon durch das Repository vorgegeben.

9 Aufruf der main-Methode in Start.java

Im Repository finden Sie ein Package namens `start`. Darin befindet sich die Klasse `Start.java`. Wir werden hier explizit die `main`-Methode zum Prüfen Ihrer Lösung aufrufen. Sie dürfen den Inhalt der Klasse `Start.java` nach Belieben - solange Sie sich an die Vorgaben halten - anpassen.

9.1 Das Fenster

(1) Bei Aufruf der `Start.java` wird ein Fenster angezeigt, das in etwa 600 × 600 (Breite mal Höhe) Pixel groß ist und in der Mitte des Bildschirms angezeigt wird.

(2) Implementieren Sie die Oberfläche responsiv. Setzen Sie eine Minimalgröße fest. Diese Minimalgröße darf höchstens 600 mal 600 groß sein. Wenn das Fenster sich zum ersten Mal öffnet, sollte es sich auf 600 mal 600 öffnen. Aufgrund der Responsitivität muss die Größe des Fensters verstellbar sein. Die einzelnen Komponenten sollten beim Vergrößern des Fenstern weiterhin fehlerfrei, vollständig und an einer sinnvollen Stelle angezeigt werden und zudem bedienbar sein. Achtung: Bei einigen Betriebssystemen wird über eine Voreinstellung der Bildschirm skaliert. Das heißt, dass die angezeigten Elemente eines Bildschirms z.B. auf 150 Prozent skaliert werden. Das können Sie in den Einstellungen entsprechend auf 100 Prozent anpassen. Beachten Sie, dass wir Ihre Abgabe auf einem nicht skalierenden Bildschirm (also auf 100 Prozent) betrachten werden.

(3) Das Programm endet, sobald S das Fenster schließt.

(4) Der Inhalt des Fensters wird in zwei verschiedene Bereiche aufgeteilt: Am rechten Rand befindet sich die *Menütafel*, die stets lesbar und bedienbar bleiben muss. Daneben befindet sich die *Anzeigetafel*, welche den restlichen Platz im Fenster einnimmt. Die hier enthaltenen Elemente sollten ebenfalls klar erkennbar und bedienbar sein.

(5) Die Elemente der Menütafel werden in 9.2 beschrieben, die Elemente der Anzeigetafel werden in 9.3 beschrieben.

9.2 Die Elemente in der Menütafel

In der Menütafel muss Folgendes vorhanden, bedienbar und einstellbar sein. Die Umsetzung ist Ihnen frei gelassen, dies kann beispielsweise durch Buttons, Eingabemöglichkeiten für S, Dropdown-Menüs, etc. erfolgen, es sei denn, etwas anderes ist bereits durch uns festgelegt. Sorgen Sie unbedingt dafür, dass wir nachvollziehen können, an welcher Stelle welche Einstellungen vorgenommen werden können (z.B. durch Beschriftung). Beispiel: Kann S irgendwo die Zahl 5 einstellen, so ist es wichtig, zu wissen, ob es sich hierbei um die Anzahl der Farben oder um die Anzahl der Spalten handelt.

(1) Es gibt die Möglichkeit, dass S sich eine Bedienungsanleitung anzeigen lassen kann. Wichtig ist hier zu wissen, WIE S spielen kann, nicht, wie die Regeln lauten. Hierbei soll also beispielsweise beschrieben werden, wie S die Farbe wählen kann (Mausklick, Ziffer, s.u.) und nicht welche Farben S wann wählen darf.

(2) Es gibt einen Button, der mit *Start* beschriftet ist. Wird dieser betätigt, werden alle Einstellungen aus der Menütafel übernommen, ein neues, zufällig gefärbtes (siehe (3)) Spielbrett generiert und angezeigt. Durch die Einstellungsmöglichkeiten, die wir hier vorgegeben haben, ist es stets möglich, ein Spielbrett zu generieren, das *startklar* ist. Gleichzeitig wird durch die Betätigung von *Start* die Beschriftung des Buttons auf *Stop* geändert. Wird dann dieser *Stop*-Button betätigt, bricht das Spiel ab, das Spielbrett verschwindet und die Beschriftung ändert sich wieder auf *Start*. Die vorgenommenen Einstellungen auf der Menütafel bleiben erhalten. Dieser Button kann beliebig oft hintereinander betätigt werden.

(3) Mit zufällig gefärbtes Spielbrett meinen wir, dass jedes Mal, wenn ein Spielbrett neu generiert und angezeigt wird, es tatsächlich zufällig neu gefärbt wird. Ein Spielbrett mit fest einprogrammierten Farben führt also zu hohem Punktverlust. Starten wir also das Spiel mehrere Male, darf es zum Beispiel nicht vorkommen, dass es genau mit den gleichen Farben auf den gleichen Feldern startet. Einzige Ausnahme dieser Regel besteht darin, dass es durch die eventuell durch die Einstellungen beschränkte Anzahl an unterschiedlichen Möglichkeiten dazu kommen kann, dass die Einfärbung des Spielbretts sich wiederholen kann.

(4) Es gibt einen Button, der mit *Play* beschriftet ist. Wird dieser betätigt, wechselt seine Beschriftung auf *Pause*, die Einstellungsmöglichkeiten werden ausgegraut und sind nicht mehr verstellbar, das Spiel beginnt mit dem aktuell angezeigten Brett und den aktuell eingegebenen Einstellungen. Wird der *Pause*-Button betätigt, wechselt seine Beschriftung auf *Play*, das Spiel wird eingefroren (insbesondere der Timer, siehe (9)) und es kann erst

dann weiter gespielt werden, wenn erneut auf *Play* gedrückt wurde. Dieser Button kann beliebig oft betätigt werden.

(5) S kann einstellen, ob S_1 oder S_2 beginnt (Standard: S_1). Es sollte stets während des Spiels auf der Menütafel erkennbar sein, wer angefangen hat.

(6) S kann einstellen, wie viele Farben im Spiel enthalten sind (alle Möglichkeiten, beginnend bei 4 bis inklusive 9, sollten wählbar sein) (Standard: 5). Es sollte stets während eines Spiels auf der Menütafel erkennbar sein, mit wie viele Farben gerade insgesamt gespielt wird.

(7) S kann einstellen, wie viele Zeilen (mindestens 3, höchstens 10) und wie viele Spalten (mindestens 3, höchstens 10) das Spielbrett enthält. Das Spielbrett kann also auch nicht-quadratisch sein. Standardgröße ist 6 Zeilen mal 6 Spalten. Es sollte stets während eines Spiels auf der Menütafel erkennbar sein, wie viele Zeilen und wie viele Spalten das Spielbrett gerade hat.

(8) S kann einstellen, welche Strategie der PC spielen soll (Strategie01, Strategie02, Strategie03; Standard: Strategie01). Es sollte stets während eines Spiels auf der Menütafel erkennbar sein, welche Strategie der PC gerade spielt.

(9) Sobald ein Spiel beginnt (durch Betätigen von *Play*) startet ein Timer, welcher stets anzeigt wird (wo, ist Ihnen überlassen) und angibt, wie lange bereits gespielt wird. Dieser Timer pausiert, wenn auf *Pause* gedrückt wird, und startet wieder, sobald auf *Play* gedrückt wird.

(10) Wird die `main`-Methode in `Start.java` aufgerufen, so sind die Standardeinstellungen bereits voreingestellt. Drückt S gleich darauf auf *Start*, so wird ein Spielbrett mit eben diesen Einstellungen angezeigt. Drückt S dann auf *Play*, startet das Spiel mit eben diesen Einstellungen und dem angezeigten Brett.

(11) Während eines Spiels wird stets angezeigt, wie groß die Komponenten von S_1 und S_2 jeweils sind.

(12) Sie dürfen in der Menütafel weitere Elemente ergänzen, wenn Sie möchten, auch das Design innerhalb der Vorgaben ist frei wählbar. Sie sollten allerdings dafür sorgen, dass nachvollziehbar ist, was gewählt werden kann und was zur Zeit gewählt ist.

9.3 Die Anzeigetafel

(1) Zu Beginn, also noch bevor S das erste Mal auf *Start* drückt, ist die Anzeigetafel leer.

(2) Auf der Anzeigetafel ist, sobald das erste Mal auf den *Start* gedrückt wurde, immer ein Spielbrett zu sehen, mindestens bis zu dem Zeitpunkt, zu dem das Spiel als beendet

gilt. Unterhalb des Spielbretts sind mit erkennbarem Abstand die auswählbaren Farben angezeigt. Zu jeder Farbe wird eine entsprechende Ziffer angezeigt, die für die Farbe steht. Dabei dürfen alle Ziffern (= einstellige Zahlen) außer der 0 verwendet werden. **(0612a)** Wie Sie die Zahlen unterhalb des Spielbretts anzeigen, überlassen wir Ihnen, siehe Hinweise. **(0612e 11.11)**.

(3) Drückt S mit der Maus auf ein Feld im Spielbrett mit einer bestimmten Farbe, oder auf eine der angezeigten Farben unterhalb des Spielbretts, dann bedeutet das, dass S eben diese Farbe als den nächsten Zug festlegt. Durch diese Wahl ändert sich das Spielbrett entsprechend: Die **(0612a)** ~~Farbfläche~~ Komponente **(0612e, siehe 11.7)** von S nimmt die neue Farbe an, inklusive der ggf. neu hinzugewonnenen Kästchen.

(4) Drückt S auf eine Ziffer auf der Tastatur (z.B. 5) und steht 5 für die Farbe Blau, so bedeutet das, dass S als nächsten Zug die Farbe Blau wählt.

(5) Gilt das Spiel noch nicht als beendet, ist im Anschluss der PC an der Reihe und wählt eine Farbe. Sorgen Sie dafür, dass eine Sekunde vergeht, bis tatsächlich die neu gewählte Farbe angezeigt wird und S wieder an der Reihe ist.

(6) Sobald das Spiel als beendet gilt, wird ein entsprechender Hinweis angezeigt. Gibt es eine:n Gewinner:in, so wird sie:er angegeben; auch bei unentschieden wird ein entsprechender Hinweis gegeben. Der Button, der mit *Stop* beschriftet ist, wechselt seine Beschriftung wieder zu *Start*. Die Einstellungen sind nicht mehr ausgegraut, zeigen aber noch die Einstellungen, die das gerade beendete Spiel hatte. Es ist nun zum Beispiel möglich, durch Drücken auf *Start* ein neues Spielbrett mit eben diesen Einstellungen anzeigen zu lassen und damit ein neues Spiel zu beginnen. Auch ist es möglich, die Einstellungen zu ändern, und dann mittels *Start* ein neues Spielbrett, entsprechend der neuen Einstellungen, anzeigen zu lassen, auf welchem ein neues Spiel begonnen werden kann.

10 Prüfen von Testing.java

Wir werden einen Teil Ihres Projekts automatisiert testen. Zu diesem Zweck gibt es die Klasse **`Testing.java`**, welche wir überprüfen werden. Dazu werden wir ein Objekt namens **`test`** von dieser Klasse erstellen. Über dieses Objekt werden wir dann die Methoden, mit den entsprechenden Übergabeparametern, aufrufen und prüfen. **(0612a)** Ergänzung: Sie dürfen davon ausgehen, dass wir die Methoden aus den Abschnitten ab Abschnitt 10.3 nur mit Spielbrettern testen werden, die vollständig initialisiert sind: mit `null` oder mit einem leeren Spielbrett werden wir also nicht testen. **(0612e, siehe 11.3)**

10.1 Technische Vorgaben - der Testing-Konstruktor

Sie müssen sich an die technischen Vorgaben halten. Weiterhin gilt auch hier, dass weitere Importe aus Ihrem Projekt / der Standardbibliothek erlaubt sind und dass Sie die **`Testing`**-Klasse mit weiteren Attributen und Hilfsmethoden versehen dürfen.
Hier gilt noch eine wichtige, weitere technische Information, die Sie zwingend berücksichtigen müssen, um überhaupt Punkte erhalten zu können: Sie dürfen zwar **weitere Konstruktoren** von **`Testing`** hinzufügen. Allerdings werden wir ausschließlich den Konstruktor, den wir Ihnen vorgeben, verwenden, um Ihr Programm zu testen. Den Rumpf des von uns vorgegebenen Konstruktors dürfen Sie aber nach Belieben anpassen (und da zum Beispiel andere Konstruktoren aufrufen).

10.2 Übergabeparameter des Testing-Konstruktors

Um die Methoden beschreiben zu können, gehen wir nun davon aus, dass **`test`** ein Objekt der Klasse **`Testing`** ist. Dabei wurde **`test`** durch den oben beschriebenen, von uns vorgegebenen Konstruktor, dessen Rumpf Sie noch anpassen dürfen, erzeugt. Dieser Konstruktor (siehe Repository) hat als Übergabeparameter ein zweidimensionales **`Field`**-Array namens **`initBoard`**. Der Konstruktor weist den Übergabeparameter an das vorhandene Objektattribut **`board`** zu (bereits im Repository umgesetzt). Bei **`initBoard`** gilt Folgendes:

(1) Alle **`Field`**-Objekte in **`initBoard`** sind bereits initialisiert.

(2) Alle **`Field`**-Objekte wurden mit der importierten **`Field`**-Klasse aus **Ihrem** Projekt erstellt (siehe Repository). Sie werden mit dem **`Field(int row, int col, int color)`**-Konstruktor erzeugt, den wir Ihnen in der **`Field`**-Klasse vorgegeben haben.

- Sie dürfen die Signatur dieses Konstruktors auf keinen Fall verändern.
- Sie dürfen den Rumpf des Konstruktors um weitere Codezeilen ergänzen. Vorhandener Code darf aber nicht gelöscht werden.
- Sie dürfen bei Bedarf weitere Konstruktoren zur **`Field`**-Klasse hinzufügen.

(3) Bei jedem **Field**-Objekt in **initBoard** sind genau drei Objektattribute gesetzt:

- die Farbe, in Form einer **int**-Zahl, gespeichert in **color**,
- der Zeilenindex, in Form einer **int**-Zahl, gespeichert in **row**,
- der Spaltenindex, in Form einer **int**-Zahl, gespeichert in **col**.

(4) Ein **Field**-Objekt in **initBoard** befindet sich genau dann in Zeile **x** und Spalte **y**, wenn der Wert in **row** den Wert **x** und der Wert in **col** den Wert **y** hat. Dabei zählen wir die erste Spalte im Spielbrett als die 0te Spalte. Analoges gilt für die Zeilenzählung.

Beachten Sie, dass **test** ein Attribut namens **board** besitzt (siehe Repository). Bei **board** handelt es sich um ein zweidimensionales **Field**-Array.
Wir möchten darauf hinweisen, dass sich aus **board** allerlei Informationen extrahieren lassen, wie zum Beispiel welche Farbe und Größe gerade die Komponenten von S_1 (zu Erinnerung: Startfeld links unten) und S_2 (zur Erinnerung: Startfeld rechts oben) haben. Jedes Spiel, das wir bei **Testing** prüfen werden, wird ein Spiel mit sechs Farben sein. Diese sechs Farben werden wir mit 1,2,3,4,5 und 6 bezeichnen. Das bedeutet nicht, dass sich in jedem Spielbrett, das wir zum Testen verwenden werden, stets alle 6 Farben befinden.

10.3 Die Methode isStartklar

Implementieren Sie die Methode **isStartklar**.

- Modifikatoren: **public**
- Rückgabetyp: **boolean**
- Name: **isStartklar**
- Übergabeparameter: **keine**

Der Rückgabewert dieser Methode ist genau dann **true**, wenn es sich bei **board** um ein Spielbrett handelt, welches *startklar* ist (siehe 4.8).

10.4 Die Methode isEndConfig

Implementieren Sie die Methode **isEndConfig**.

- Modifikatoren: **public**
- Rückgabetyp: **boolean**
- Name: **isEndConfig**
- Übergabeparameter: **keine**

Der Rückgabewert dieser Methode ist genau dann **true**, wenn es sich bei **board** um ein Spielbrett handelt, welches einer Endkonfiguration entspricht (siehe 4.9).

10.5 Die Methoden testStrategy01, testStrategy02, testStrategy03

Sei $k \in \{1, 2, 3\}$. Ersetzen Sie in dieser Beschreibung k mit dem entsprechenden Wert und implementieren Sie die entsprechende Methode.

- Modifikatoren: **`public`**
- Rückgabetyp: **`int`**
- Name: **`testStrategy0k`**
- Übergabeparameter: **`keine`**

Diese Methode gibt eine Zahl zurück, die für die Farbe steht, die S_2 als nächstes wählt, unter der Bedingung, den nächsten Zug entsprechend Strategie0k zu wählen.

10.6 Die Methode toBoard

Implementieren Sie die folgende Methode.

- Modifikatoren: **`public`**
- Rückgabetyp: **`boolean`**
- Name: **`toBoard`**
- Übergabeparameter: **`Field[][] anotherBoard, int moves`**

Diese Methode gibt genau dann **`true`** zurück, wenn Folgendes erfüllt ist:

(1) S_1 ist an der Reihe (**(0612a)** davon dürfen Sie ausgehen, siehe Hinweise **(0612e, siehe 11.12)**),

(2) es ist möglich, innerhalb von höchstens **`moves`** vielen Zügen **`board`** so zu ändern, dass die Farbverteilung identisch zu der von **`anotherBoard`** ist.

Beispiele / Hinweise:

- Beispiel: Es ist möglich, vom Spielbrett in Abb. 1 zum Spielbrett in Abb. 2 in genau 4 Zügen zu gelangen (S_1: Blau, S_2: Rot, S_1: Lila, S_2: Gelb). Weiterhin ist es nicht möglich, in weniger Zügen diese Umwandlung zu schaffen. Da es in genau 4 Zügen möglich ist, kann die Frage, ob es auch in **höchstens** 5 oder 6 oder ... Zügen möglich ist, mit ja beantwortet werden. Folglich gilt, dass wenn in **`moves`**
  - der Wert 0 oder 1 oder 2 oder 3 gespeichert ist, **`false`** zurück gegeben wird.
  - der Wert 4 oder größer gespeichert ist, **`true`** zurück gegeben wird.
- Hinweis 1: Sie dürfen davon ausgehen, dass der Wert, der in **`moves`** gespeichert ist, mindestens 0 ist.
- Hinweis 2: **`moves`** entspricht der Anzahl der Züge von S_1 und S_2 zusammen.

10.7 Die Methode minMoves

Hier spielt S_1 das Spiel alleine, das heißt, nach jedem Zug von S_1 ist S_1 wieder dran und darf auch die Farbe wählen, die das Feld oben rechts hat.
Implementieren Sie die folgende Methode.

- Modifikatoren: **`public`**
- Rückgabetyp: **`int`**
- Name: **`minMoves`**
- Übergabeparameter: **`int x, int y`**

Diese berechnet die Anzahl an Zügen, die S_1 mindestens braucht, um das **`Field`**-Objekt mit Zeilenindex **`x`** und Spaltenindex **`y`** einzunehmen (also zum Teil der eigenen Komponente zu machen), ausgehend von dem in **`board`** gespeicherten Spielbrett. Ist das entsprechende **`Field`**-Objekt bereits eingenommen, so wird der Wert 0 zurück gegeben. Bei der Wahl der Farben muss S_1 aufsteigend und zyklisch vorgehen (siehe **`minMovesFull`**).

10.8 Die Methode minMovesFull

Hier spielt S_1 das Spiel alleine, das heißt, nach jedem Zug von S_1 ist S_1 wieder dran und darf auch die Farbe wählen, die das Feld oben rechts hat. Implementieren Sie die folgende Methode.

- Modifikatoren: **`public`**
- Rückgabetyp: **`int`**
- Name: **`minMovesFull`**
- Übergabeparameter: **`keine`**

Diese berechnet die Anzahl an Zügen, die S_1 mindestens braucht, um das gesamte Spielbrett in einer Farbe zu färben. Dabei darf S_1 die Farben nur zyklisch und aufsteigend sortiert wählen, wobei allerdings nicht festgelegt ist, mit welcher Farbe S_1 beginnt: S_1 wählt z.B. als erste Farbe die Farbe 1, muss aber dann 2, dann 3, bis zur 6 wählen, und beginnt dann wieder von vorne bei 1, dann 2, und so weiter. Die erste Farbe von S_1 könnte aber z.B. auch die Farbe 3 sein. Die nächste muss dann 4, 5, 6 und anschließend wieder 1,2,3,4,5,6 und wieder 1, 2, und so weiter sein, bis das Spielbrett eine Farbe hat. Unter diesen zylkischen Reihenfolgen ermittelt die Methode also die kleinste Anzahl an Zügen, die notwendig ist, um das Spielbrett in einer Farbe zu färben.
Ist das Spielbrett bereits einfarbig gefärbt, so wird der Wert 0 zurück gegeben.

11 Hinweise und Anpassungen

Wir danken Ihnen für die Fragen und Hinweise, die uns erreichen. Hier eine Übersicht der Anpassungen, die wir in diesem Dokument vorgenommen haben. Ein paar Hinweise / Antworten haben Sie ja bereits persönlich in den Fragestunden oder über Ilias erhalten.

11.1 Abschnitt 4.4 - zusammenhängend

Hier wurde eine genauere Formulierung vorgenommen, siehe 4.4.

11.2 Abschnitt 4.10 - Wahl einer Farbe

Hier wurde eine Formulierung präzisiert, siehe 4.10. Wir möchten darauf hinweisen, dass es ein paar Studierende gab, die in der Fragestunde nachgefragt haben und eine anderslautende Antwort erhalten haben. Wir bitten, dies zu entschuldigen, es basierte auf einem Kommunikationsproblem im Team. Es gilt das, was hier steht.

11.3 Abschnitt 10 - Testen von isStartklar und andere

Es gab Verunsicherung, inwiefern `null` u.ä. abgefangen werden muss. Dies haben wir nun klargestellt. Siehe 10. Wurd am 09.06. auf Ilias bekannt gegeben.

11.4 Abschnitt 8 - Strategien aus Sicht von S_2

In Abschnitt 10.5 wird verlangt, dass die Strategien aus Sicht von S_2 implementiert werden müssen. In Abschnitt 8 werden die Strategien aus Sicht von S_1 und S_2 beschrieben. Dies hat zu Verunsicherung geführt. Daher haben wir *aus Sicht von* S_2 noch in Abschnitt 8 hinzugefügt.

11.5 Abschnitt 4.9 - Endkonfiguration

Wir mussten die Definition einer Endkonfiguration anpassen, da wir einige Fälle bei der alten, vereinfachten Definition nicht bedacht hatten und daher die bisherige Beschreibung nicht konsistent war. Siehe 4.9.

11.6 Abschnitt 8.3 - Wort Farbfläche ersetzt I

Farbfläche ist synonym zu Komponente. Um konsistent zu bleiben, haben wir Farbfläche mit dem Wort Komponente ersetzt. Siehe 8.3.

11.7 Abschnitt 9.3 - Wort Farbfläche ersetzt II

Farbfläche ist synonym zu Komponente. Um konsistent zu bleiben, haben wir Farbfläche mit dem Wort Komponente ersetzt. Siehe 9.3.

11.8 Abschnitte 10.6, 10.7 und 10.8

Formatierungsfehler. Zuvor waren es Unterabschnitte. Nun sind es „normale“ Abschnitte.

11.9 Abschnitt 8 - Testen mit gutartigen Spielbrettern

Insbesondere gilt also für jedes Feld im Spielbrett, welches wir zum Testen der Strategien verwenden werden, genau eine der folgenden Eigenschaften:

- Das Feld gehört zu einer Komponente.
- Das Feld grenzt an mindestens eine Komponente (gehört aber zu keiner Komponente). Sei M die Menge an Feldern, die gebildet wird durch das Feld selbst und all seine Nachbarn, die nicht Teil einer Komponente sind. Dann sind die Felder in M paarweise verschieden gefärbt.
- Das Feld grenzt an keine Komponente (und ist auch nicht Teil einer Komponente). Sei M die Menge an Feldern, die gebildet wird durch das Feld selbst und all seine Nachbarn. Dann sind die Felder in M jeweils paarweise verschieden gefärbt.

11.10 Abschnitt 8.1 - falsche Bezeichnung im Beispiel

Statt S_2 stand hier zuvor S_1. Dieser Fehler in dem Beispiel wurde am 30.5. auf Ilias bekannt gegeben.

11.11 Abschnitt 9.3

Ergänzung zu der Anzeige der Farben unter dem Spielbrett: Bei den angezeigten Farben ist Ihnen freigestellt, ob Sie auch wirklich nur die Farben anzeigen, die die Spieler (oder der PC) gerade auswählen können oder ob Sie zwar alle Farben zeigen, aber klar kenntlich machen, welche die auswählbaren Farben sind (durch ausgrauen von Farben, verschiedenen Größen, etc.). Hauptsache ist, dass wir den Unterschied ganz klar erkennen können. Wurde am 07. Juni auf Ilias bekannt gegeben.

11.12 Abschnitt 10.6 - S1 ist an der Reihe

Ergänzung zur Methode `toBoard`: Die Bedingung "(1) S1 ist an der Reihe" ist so zu verstehen, dass Sie davon ausgehen dürfen, dass S1 den ersten Zug macht. Wurde am 09.06. auf Ilias veröffentlicht.