IFM 2.1 Programmiermethoden, S20

B02 'Tic Tac Toe': Strategie-Pattern, UML, JavaDoc, Git Branching

Praktikum: 22.05.2020
ILIAS: 21.05.2020

Carsten Gips, FH Bielefeld

1 Aufgabe Praktikum

1.1 Strategie-Entwurfsmuster

Implementieren Sie das Spiel "Tic Tac Toe" (Spielregeln vergleiche wikipedia.org/wiki/Tic_Tac_Toe) in Java. Nutzen Sie dazu die Vorgaben (Interfaces und Packages): Forken/Clonen Sie das Vorgabe-Repo unter git01-ifm-min.ad.fh-bielefeld.de/Lehre/pm-s20/pm-vorgaben-s20 und arbeiten Sie Ihrem Clone weiter.

Es soll eine eigene Klasse für die Spieler geben, die das vorgegebene Interface `ttt.game.IPlayer` implementiert. Ihr Spiel muss das vorgegebene Interface `ttt.game.IGame` implementieren. Nutzen Sie für die Repräsentation der Spielzüge das Interface `ttt.game.IMove` aus der Vorgabe.

Implementieren Sie das Strategy-Pattern, um den Spielerinstanzen zur Laufzeit eine konkrete Spielstrategie mit dem Interface `ttt.strategy.IGameStrategy` mitzugeben, nach denen die Spieler ihre Züge *berechnen*. Implementieren Sie mindestens drei unterschiedliche konkrete Strategien.[1]

Nutzen Sie als weitere Strategie den stets perfekt spielenden MinMax-Algorithmus[2]. Der Algorithmus steht über die Klasse `ttt.strategy.MinMaxStrategy` im `.jar`-File in den Vorgaben zur Verfügung (die Klasse implementiert das Interface `ttt.strategy.IGameStrategy`).

Hinweis: Ein Spieler soll in `IPlayer#nextMove` seinen nächsten Zug nur **berechnen** und darf nicht direkt den Spielstand (Spielbrett) modifizieren! Das eigentliche Durchführen des berechneten Zuges geschieht über die Methode `IGame.doMove()` aus dem Spiel. Die Berechnung im Spieler wird an die zur Laufzeit übergebene Strategie delegiert.

Hinweis: Binden Sie das `.jar`-File mit dem MinMax-Algorithmus so ein, dass es zum Compilieren und zum Starten Ihres Programms genutzt wird. Sie sollen die Datei nicht entpacken!

Gehen Sie bei der Lösung der Aufgabe methodisch vor:

a) Stellen Sie sich eine Liste mit relevanten Anforderungen zusammen.

b) Erstellen Sie (von Hand) ein Modell (UML-Klassendiagramm):

- Welche Klassen und Interfaces werden benötigt?
- Welche Aufgaben sollen die Klassen haben? Welche Attribute und Methoden sind nötig?
- Wie sollen die Klassen interagieren, wer hängt von wem ab?

c) Implementieren Sie Ihr Modell in Java. Schreiben Sie ein Hauptprogramm, welches das Spiel startet und die Spieler abwechselnd ziehen lässt.

d) Überlegen Sie, wie Sie Ihr Programm sinnvoll manuell (noch ohne JUnit) testen können und tun Sie das.

e) Beachten Sie die Style- und Namensrichtlinien und dokumentieren Sie Ihr Programm mit Javadoc (vgl. B01).

Thema: Objektorientierter Entwurf und Implementierung, manuelles Testen, Nutzung des Strategie-Entwurfsmusters

[1] Eine mögliche Strategie könnte sein, den Nutzer via Tastatureingabe nach dem nächsten Zug zu fragen.
[2] Auf die Funktionsweise des MinMax-Algorithmus wird im Wahlmodul "Künstliche Intelligenz" genauer eingegangen :-)

1.2 Charts mit Java

Laden Sie sich von github.com/jfree/jfreechart die JFreeChart-Bibliothek für Java herunter und binden Sie diese in Ihr Projekt ein.[3] Erzeugen Sie mit Hilfe dieser Bibliothek mindestens drei verschiedene sinnvolle Charts (d.h. drei unterschiedliche Diagrammtypen) zum Überblick über die Spiele und/oder Spieler.

Achtung: Die Dokumentation zu JFreeChart ist kostenpflichtig! Sie benötigen diese aber nicht — nutzen Sie stattdessen die kostenfrei verfügbare JavaDoc-Dokumentation der API sowie die FAQ.

Thema: Einarbeitung in fremde APIs, Erstellen von Diagrammen in Java

2 Tutorium

Lösen Sie die folgenden Aufgaben selbstständig **vor** dem Tutorium und diskutieren Sie Ihre Lösung und ggf. aufgetauchte Probleme und Fragen im Tutorium.

Hinweis: Beachten Sie dabei auch den Zeitplan der Themen in der Vorlesung: Sie werden entsprechend nicht alle Fragen gleich zum ersten Tutorium lösen können!

2.1 Vererbung und dynamische Polymorphie

Erklären Sie folgenden Code. Welche Ausgabe würde man jeweils erhalten, wenn man `a1`, `a2`, `a3` und `b1` mit Hilfe von `System.out.println()` ausgeben würde? Begründen Sie Ihre Antwort!

```
class A {
    int val = 0;

    A() { this(42); }
    A(int v) { val = v; }
    public String toString() { return String.valueOf(val); }
}
class B extends A {
    int val = 10;

    public static void main(String[] args) {
        A a1 = new A();    A a2 = new A(3);    A a3 = new B();    B b1 = new B();
    }
}
```

Was würde sich ändern, wenn die Klasse `B` den folgenden Konstruktor hätte? Begründen Sie Ihre Antwort!

```
    B() { super(); val = 9; }
```

Was würde sich ändern, wenn die Klasse `B` zusätzlich folgende Methode hätte? Begründen Sie Ihre Antwort!

```
    public String toString(int val) { return String.valueOf(val); }
```

Thema: Sicherer Umgang mit Vererbung und dynamischer Polymorphie (Wiederholung, Vertiefung)

[3] Sie finden unter repo1.maven.org/maven2/org/jfree/jfreechart/1.5.0/jfreechart-1.5.0.jar die kompilierte Bibliothek, nutzen Sie diese!

2.2 Git Branches und Mergen – Kommandozeile

Üben Sie den Umgang mit Git auf der Kommandozeile:

a) Legen Sie in Ihrem Projekt einen Branch an. Ändern Sie einige Dateien und committen Sie die Änderungen. Checken Sie den Master-Branch aus und mergen Sie die Änderungen. Was beobachten Sie?

b) Legen Sie einen weiteren Branch an. Ändern Sie einige Dateien und committen Sie die Änderungen. Checken Sie den Master-Branch aus und ändern Sie dort ebenfalls:

- Ändern Sie eine Datei an einer Stelle, die nicht bereits im Branch modifiziert wurde.
- Ändern Sie eine Datei an einer Stelle, die bereits im Branch manipuliert wurde.

Committen Sie die Änderungen.

Mergen Sie den Branch jetzt in den Master-Branch. Was beobachten Sie? Wie lösen Sie Konflikte auf?

Thema: Sicherer Umgang mit den grundlegenden Arbeitsabläufen in Git

2.3 Git: Branching-Strategien und Workflows

Worin unterscheiden sich die Branching-Strategien "Git-Flow-Modell" und "GitHub-Flow-Modell"? Wo liegen jeweils die Vor- und Nachteile aus Ihrer Sicht?

Welche Aufgaben hat ein Integrationsmanager in einem (zentralisierten) Workflow mit Integrationsmanager oder im "Dictator and Lieutenants Workflow"? Wie gestaltet sich die Mitarbeit in solchen Projekt-Workflows als "normaler" Beitragender?

Thema: Sicherer Umgang mit den grundlegenden Arbeitsabläufen in Git

2.4 Github-Workflow und Merge-Requests

Üben Sie gemeinsam das Erstellen von Merge-Requests:

- Erstellen Sie sich ein "Blessed Repo", forken Sie dieses und clonen Sie den den Fork lokal.
- Arbeiten Sie in Ihrem Clone mit Themenbranches und stellen Sie Merge-Requests für die Themenbranches in den Master-Branch in **Ihrem Fork**.
  - Achten Sie auf die Vollständigkeit des MRs: Assignee, Summary, Description!
  - Kommentieren Sie gegenseitig die Code-Stellen.
  - Nehmen Sie weitere Commits in Ihren Themenbranches vor und pushen Sie diese, damit die Änderungen Teil des Merge-Requests werden.
  - Beobachten Sie dabei, ob die Kommentare geschlossen werden.
  - Wie kann der MR akzeptiert bzw. abgelehnt werden? Welche Optionen gibt es dabei, was passiert mit den Themenbranches?
- Arbeiten Sie nun den erteilten Workflow "Forked Public Project" durch:
  - Fügen Sie das "Blessed Repo" als weiteres Remote Repo in Ihrem Clone hinzu.
  - Erstellen Sie für Themenbranches in Ihrem Clone einen MR gegen den Master-Branch im "Blessed Repo".
  - Einer im Team übernimmt die Rolle des Maintainers und kommentiert den Code. Die anderen Teammitglieder fixen ihren Code und pushen die Änderungen
  - Der Maintainer akzeptiert den MR, das lokale Repo und der Fork müssen nun aktualisiert werden.

Thema: Einüben des Github-Workflows und der Zusammenarbeit im "Forked Public Project"-Modell