Technische Universität Berlin

Software and Embedded Systems Engineering Group
Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2022/2023

Prof. Dr. Sabine Glesner
Milko Monecke
Simon Schwan

Hausaufgabenblatt UML/OCL

Ausgabe: 23.1.

Sie haben den Auftrag erhalten, ein Online-Spiel-Portal zu entwickeln. Dieses Spiel-Portal, der *GameServer*, soll flexibel für verschiedene Spiele verwendet werden können. Folgende Anforderungen wurden für das Portal definiert:

Das System soll es Benutzys[1] (*User*) ermöglichen, gegeneinander oder gegen Bots beliebige Spiele zu spielen. Dafür soll eine Server-Anwendung sämtliche Funktionalität bereit stellen, auf die Benutzys über ein Web-Interface zugreifen können. Jedes Benutzy muss sich am System registrieren bevor es sich anmelden kann, dazu muss es einen beliebigen eindeutigen Benutzynamen, einen Anzeigenamen und ein Passwort angeben. Nach dem Registrieren erfolgt eine erste Anmeldung automatisch.

Ein Benutzy kann ein neues Spiel beginnen. Dabei wählt es aus, wie viele seiner Gegenspielys durch Bots besetzt werden. Die minimale und maximale Anzahl Spielys für ein Spiel wird dabei durch das *konkrete Spiel* vorgegeben. Nach der Erstellung des Spiels werden die Bots als Spielys direkt zugewiesen, und auf menschliche Gegenspielys wird gewartet. Sobald genügend Spielys zugewiesen sind, wird das Spiel gestartet. Das Spiely, das das Spiel erstellt hat, ist zuerst am Zug.

Ein Benutzy kann außerdem einem Spiel beitreten. Wenn zu diesem Zeitpunkt bereits Spiele erstellt wurden, die noch auf Spielys warten, wird es dem Spiel, das am längsten wartet, als Spielys zugewiesen. Sollte kein Spiel auf Spielys warten, wird die Anfrage mit einer entsprechenden Meldung an das Benutzy beendet.

Ein Benutzy kann in maximal acht Spielen, die noch nicht beendet sind, als Spiely zugewiesen sein. Es kann sich jederzeit die Spiele auflisten, bei denen es mitspielt, und sich

[1] Entgendern nach Phettberg. Eine Erklärung und Begründung dazu gibt es auf der ISIS-Seite zum Kurs.

zu einem Spiel anzeigen lassen, welches der aktuelle Spielstand (Figuren auf dem Brett) ist, wer an der Reihe ist, und welchen Status das Spiel hat. Der Status kann nach außen hin folgende Werte annehmen: ,,Warten auf Spiely(s)", ,,Aktiv", ,,Beendet", ,,Beendet - Unentschieden", ,,Beendet - Aufgegeben". Intern kann dies auch detaillierter modelliert werden.

Sobald das Spiely an der Reihe ist, hat es folgende Optionen: Es kann einen Zug versuchen. Dieser Zug wird dem Spiel übermittelt, welches ihn prüft und ggf. durchführt. Sollte dieser Zug das Spiel beenden, wird das Spiel als beendet markiert und die Spielys entsprechend als Gewinny bzw. Verliery markiert. Das Spiely kann außerdem ein Unentschieden vorschlagen, welches für es für dieses Spiel vermerkt wird. Wenn alle Spielys dies getan haben, wird das Spiel als Unentschieden beendet. Ein Spiely kann außerdem aufgeben, in diesem Fall wird er zum Verliery und alle anderen Spielys haben gewonnen. Der Status ist dann ,,Beendet - Aufgegeben".

Das Benutzy kann sich jederzeit eine Statistik anzeigen lassen, die die folgenden Werte enthält:

- Anzahl gespielter Spiele (Gewonnen/Verloren/Unentschieden)
- Anteil gewonnener/verlorener/unentschiedener Spiele
- durchschnittliche Anzahl Spielzüge

Es sollen alle Spielzüge im System hinterlegt sein, um diese Statistiken jederzeit aktualisieren und den Verlauf grafisch darstellen zu können.

Das Web-Interface ermöglicht die Kommunikation des Benutzys mit dem System. Es implementiert dabei keine eigene Funktionalität, sondern reicht alle Anfragen des Benutzys direkt an das System weiter. Es kann davon ausgegangen werden, dass alle Daten, z.B. Spielstand, Züge und ggf. Textmeldungen als Antwort auf Aktionen als Strings ausgetauscht werden können.

Zu beachten ist, dass diese Modellierung nur eine abstrakte Sicht auf Spiel und Spielys enthält - das tatsächliche Prüfen und Durchführen einer Aktion im Spiel und das automatische Beenden nach einem Zug wird vom konkreten Spiel implementiert, das hier nicht beschrieben ist. Teile des Modells sind also abstrakt zu halten, damit sie später durch konkrete Instanzen implementiert werden können.

1. Use-Case-Diagramm

Modelliert die Anwendungsfälle des Systems mit einem Use-Case-Diagramm.

2. Klassendiagramm

Erstellt für das System ein Klassendiagramm, in dem alle Anforderungen bzgl. der Struktur, die aus dem Text hervorgehen, enthalten sind. Dabei ist folgendes zu beachten:

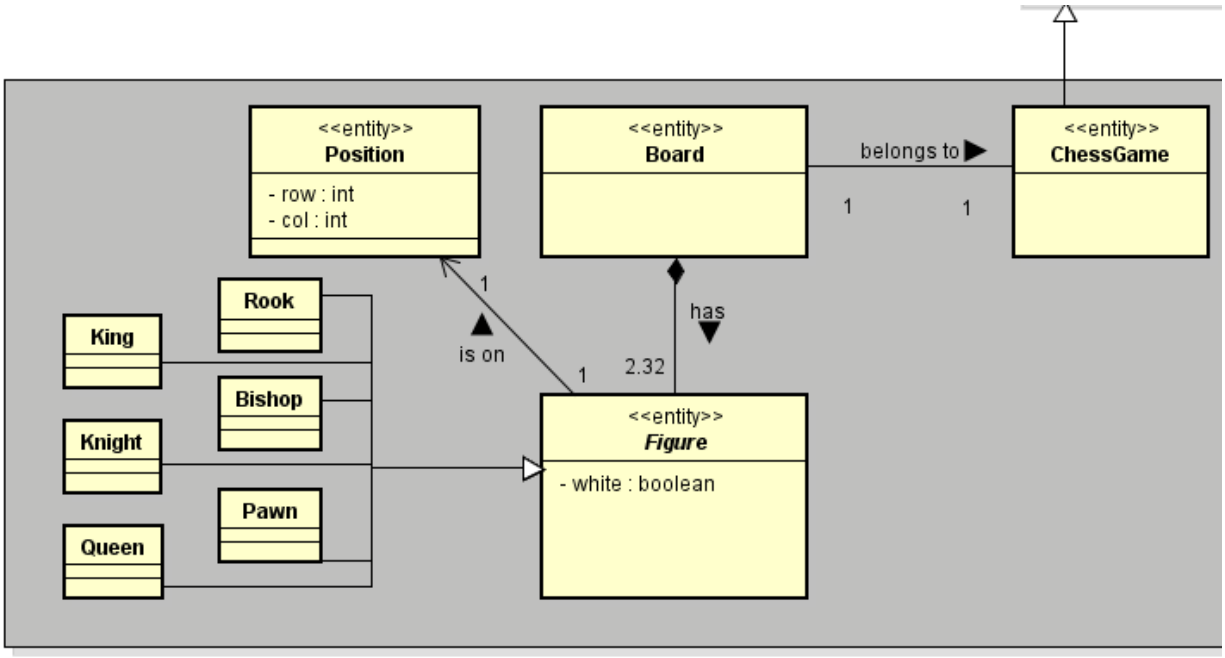
- Die in Aufgabe 1 identifizierten Use-Cases müssen als Operationen mit notwendigen Parametern auftauchen. Von weiteren Operationen kann abstrahiert werden, sofern sie nicht relevant sind.
- Assoziationen sollten über einen Namen und eine Leserichtung verfügen. Multiplizitäten und Navigationsrichtungen sind erforderlich sofern es aus den Anforderungen hervorgeht.
- Alle Klassen müssen über eindeutige Stereotype entsprechend dem Entity-Control-Boundary Pattern aus der Vorlesung verfügen.

3. Aktivitätsdiagramm

Modelliert den Ablauf (Workflow) eines Spiels in einem Aktivitätsdiagramm von seiner Erstellung über den Start bis zum Ende. Außerdem sollen die möglichen Züge der Spielys modelliert werden. Von Details eines konkreten Spiels soll abstrahiert werden, ihr dürft dafür die vorgegebenen Aktionen „Zug prüfen“ und „Zug ausführen“ verwenden. Es kann davon ausgegangen werden, dass Benutzys angemeldet sind.

4. Invarianten

Erweitert nun euer Modell um das konkrete Spiel „Schach". Beispiel für das Klassendiagramm:



Präzisiert eure Modellierung formal mithilfe von folgenden Invarianten in OCL:

a) Die Benutzys haben eindeutige Benutzykennungen.

b) Zu jedem Spiel gehört mindestens ein Spiely. Zu jedem gestarteten Schachspiel gehören zwei Spielys.

c) Kein Benutzy darf mehrere Spiely in einem Spiel kontrollieren.

d) Benutzys dürfen nicht gleichzeitig in mehr als acht Spielen, die nicht beendet sind, teilnehmen.

e) Die Anzahl der gewonnenen Spiele in der Statistik eines Benutzys entspricht den beendeten Spielen, für die dieses Benutzy tatsächlich als Gewinny vermerkt ist.

f) Bevor ein Schachspiel startet, stehen auf dem Spielfeld 16 Figuren jeder Farbe.

g) Auf einer Position dürfen niemals zwei Figuren stehen.

h) Wenn alle Spiely ein Unentschieden gefordert haben ist das Spiel als Unentschieden beendet.

i) Zu langes Warten auf Zuweisung eines Gegenspielys soll verhindert werden: Es kann kein Spiel mehr als ein menschliches Spiely haben solange ein früher erstelltes Spiel noch auf Spielys wartet.

Hinweise:

- Wir empfehlen, alle OCL-Ausdrücke mit dem USE-OCL-Tool zu überprüfen (`http://sourceforge.net/projects/useocl/`).

5. Contracts in OCL

Spezifiziert folgende Operationen mithilfe von Contracts in OCL. Berücksichtigt mögliche Fehlerfälle. Sofern dies für den Contract erforderlich ist könnt ihr davon ausgehen, dass das Benutzy durch das Web-Interface am System angemeldet wurde und die Benutzydaten als Objekt zur Verfügung stehen.

a) Benutzy registrieren

b) Aufgeben

c) Unentschieden vorschlagen

d) Spiel starten

e) Eine Funktion `calculateAverageMoves`, in der die durchschnittliche Anzahl der Züge aller beendeten Spiele eines Benutzys ermittelt wird.

6. Objektdiagramm

Stellt einen Zustand des Systems in einem Objektdiagramm dar, der entsprechend eurem Klassendiagramm und den OCL-Invarianten gültig ist. Dabei soll mindestens ein Schachspiel auf Spielys warten und eines beendet sein. Um das Diagramm übersichtlich zu halten muss Invariante 5f nicht unbedingt erfüllt sein.

7. Hoare Kalkül

Gegeben ist eine Implementierung der Operation `calculateAverageMoves` aus Aufgabe 6.e). Beweist mithilfe der Ableitungsregeln des Hoare Kalküls, dass das Programm nach Ausführung die Nachbedingung erfüllt (*partielle* Korrektheit). Führt außerdem einen Terminierungsbeweis für das Programm durch.

`calculateAverageMoves(int[] moves):`

$P\{moves.length >= 0\}$

```
sum := 0;
cnt := 0;
erg := 0;

while cnt < moves.length do
     sum := sum + moves[cnt];
     cnt := cnt + 1
od;

if moves.length >0 then
     erg := sum / cnt
else
     skip
fi
```

$$Q\{(moves.length = 0 \wedge erg = 0) \ \vee \ (moves.length > 0 \wedge erg = \frac{\sum_{i=0}^{moves.length-1} moves[i]}{moves.length})\}$$

Generelle Hinweise und Abgabeformalitäten

- Die Bearbeitung dieser Hausaufgabe ist freiwillig und bringt keine Portfoliopunkte. Es dient der Vertiefung des Stoffs im Softwaretechnik-Anteil der Veranstaltung und daher auch der Vorbereitung für den ersten Test. Die Bearbeitung ist dringend empfohlen.
- Die Bearbeitung kann in Gruppen beliebiger Größe erfolgen. Die Gruppenarbeit sollte dabei für Diskussion und Austausch genutzt werden. Sinnvoll ist es aber, jede Aufgabe auch selbst zu bearbeiten. Nur wer selbst übt, lernt es auch!
- Die Ergebnisse (auch Teilergebnisse) eurer Arbeit könnt ihr mit den Tutorys in den Tutorysprechstunden bis zum ersten Test diskutieren. Denkt daran: Direkt vor den Tests sind die Sprechstunden üblicherweise recht voll. Vorher ist also besser. Stimmt euch mit eurem Tutory ab.
- Für die Modellierung könnt ihr je nach Geschmack Deutsch oder Englisch verwenden.