Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ws2324/programming-and-modelling/ zu berücksichtigen.

Abgabefrist ist der 16.11.2023 - 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. Jedes Repository ist nur für den Studierenden selbst sowie für die Betreuer und Korrektoren sichtbar.

Für die Hausaufgabe benötigst du **ein neues** Repository.

Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/Mk5X3P_p

Nicht oder zu spät gepushte (Teil-)Abgaben werden mit 0 Punkten bewertet!

Vorbereitung

Zur Bearbeitung der Aufgabe 2 sollte eine Entwicklungsumgebung verwendet werden. Wir empfehlen aufgrund der Nachvollziehbarkeit die Verwendung von IntelliJ (siehe Anhang). Die Abgabe muss als **lauffähiges** Projekt abgegeben werden.

Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Java

Wir verwenden für die Veranstaltung Java 17.

Vorgegebenes Java-Projekt

Dein Repository enthält bereits ein Java-Projekt, das mit IntelliJ bearbeitet werden kann.

Aufgabe 1 - Klassendiagramm „Risk" (30P)

Zur Erstellung der geforderten Diagramme kann die Webanwendung "Diagrams.net" verwendet werden: https://www.diagrams.net

Gegeben sind Objektdiagramme zu Start- und Endsituationen der Szenarien aus Hausaufgabenblatt 2 (Abbildungen 1 bis 6). Erstelle, wie in der Vorlesung/Übung erläutert, **ein** Klassendiagramm, das all diese Objektdiagramme widerspiegelt. Dies bedeutet, dass das Klassendiagramm eine klare Abstraktion der Objektdiagramme darstellen muss.

Verbindungen zwischen Klassen sollen immer durch Assoziationen dargestellt werden! (Nicht als Attribute!)

Lege die erstellte PDF-Datei mit dem Namen „classdiagram.pdf" in einem Ordner mit dem Namen "task1" in deinem Repository ab. Committe und pushe die Änderung abschließend auf den main-Branch.

Bei der Bewertung wird vor allem auf die vorgestellten Konventionen der Diagrammtypen geachtet.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Handschriftliche Abgaben werden nicht gewertet.

Scenario 1: Moving Troops

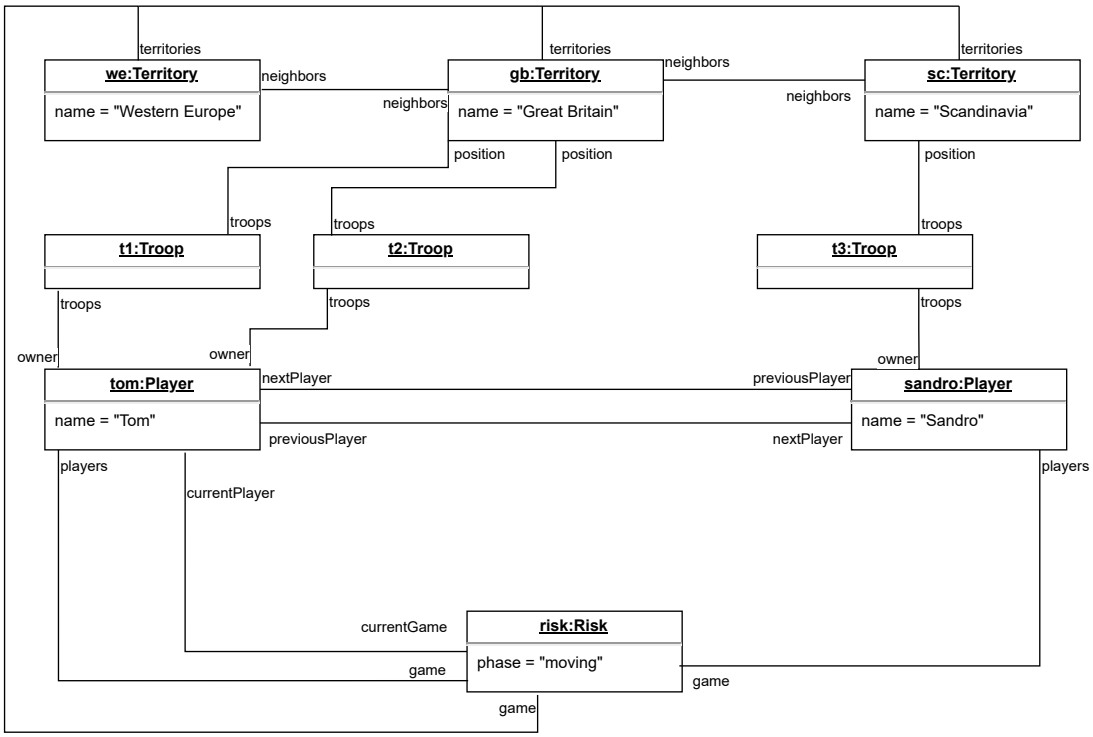

Abbildung 1: Scenario 1: Moving Troops - Start

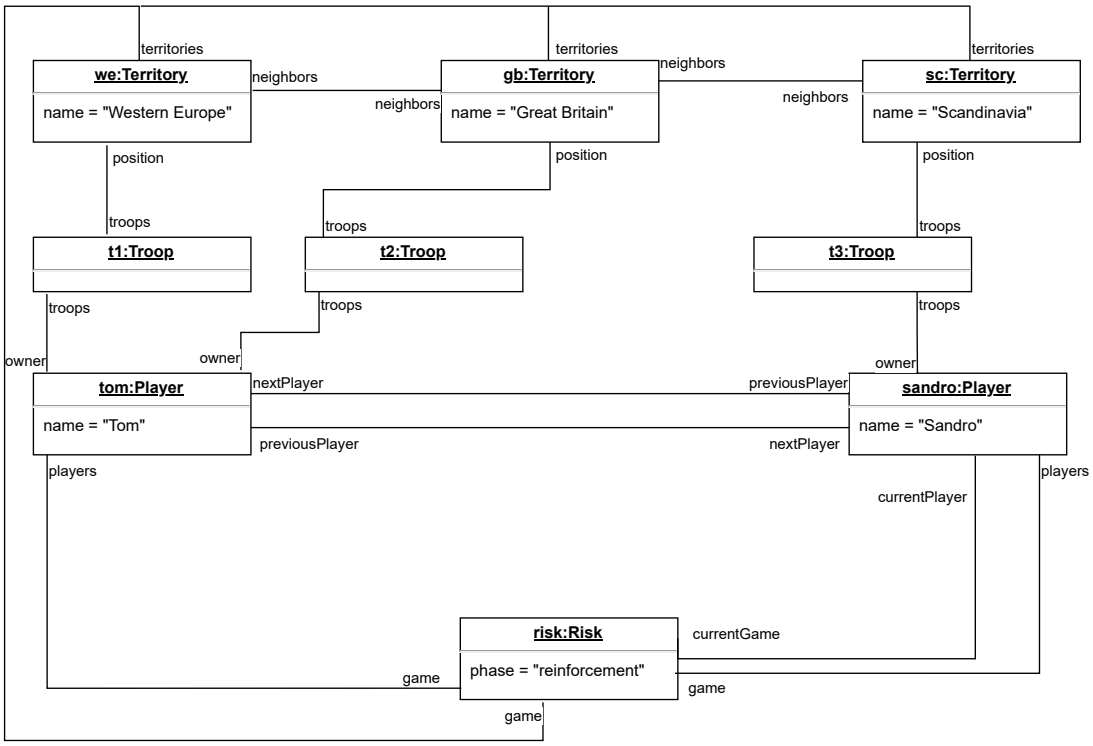

Abbildung 2: Scenario 1: Moving Troops - End

Scenario 2: Reinforcement

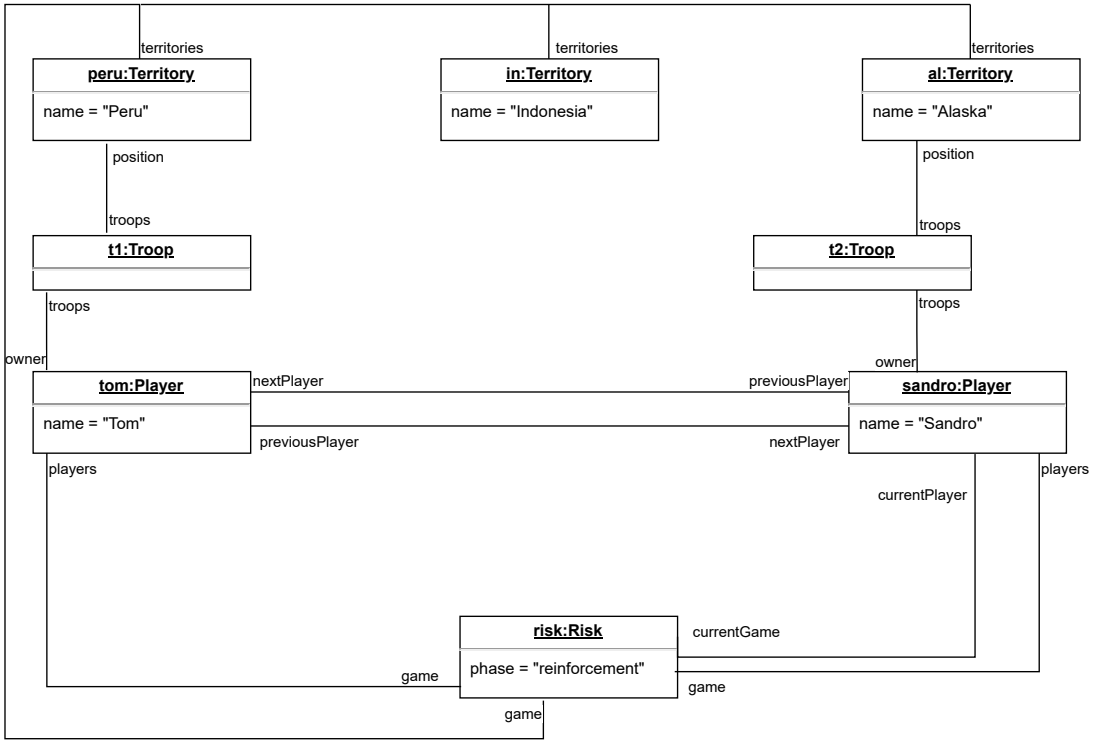

Abbildung 3: Scenario 2: Reinforcement - Start

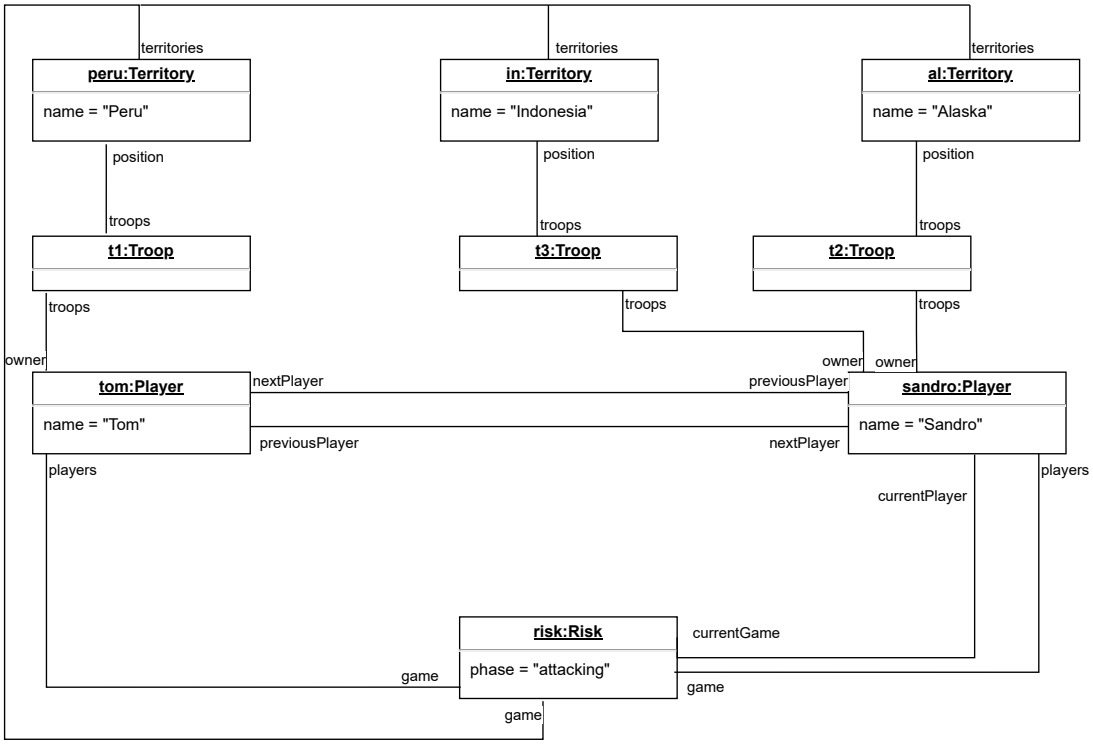

Abbildung 4: Scenario 2: Reinforcement - End

Scenario 3: Winning

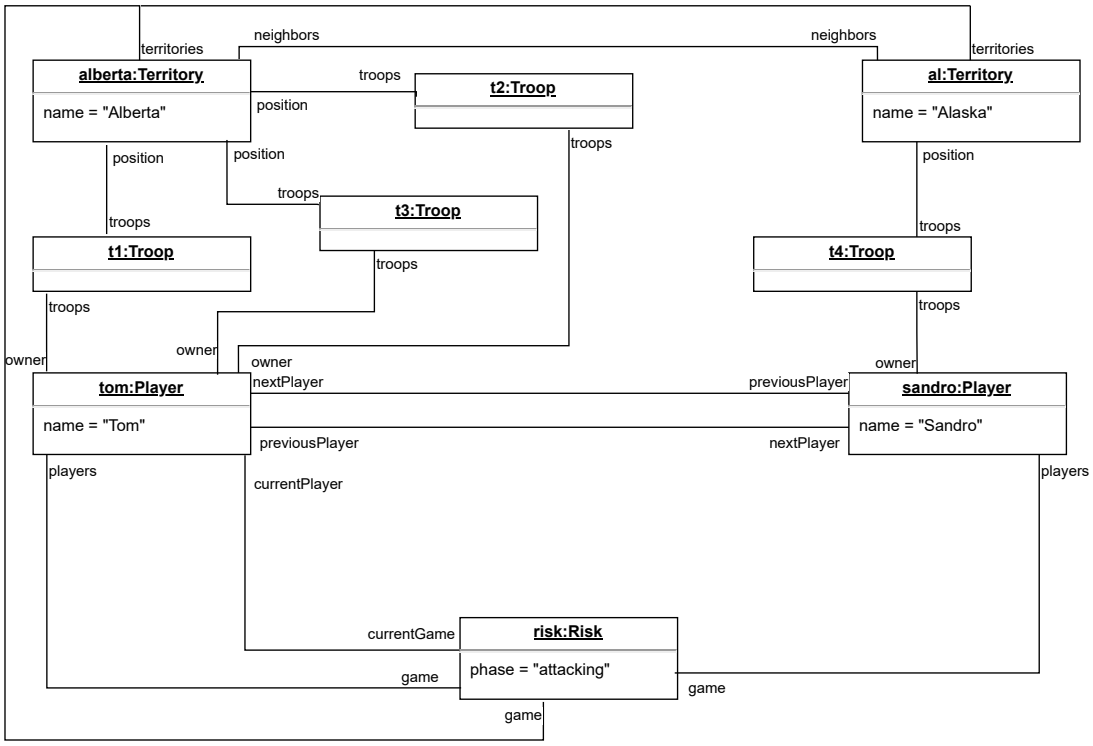

Abbildung 5: Scenario 3: Winning - Start

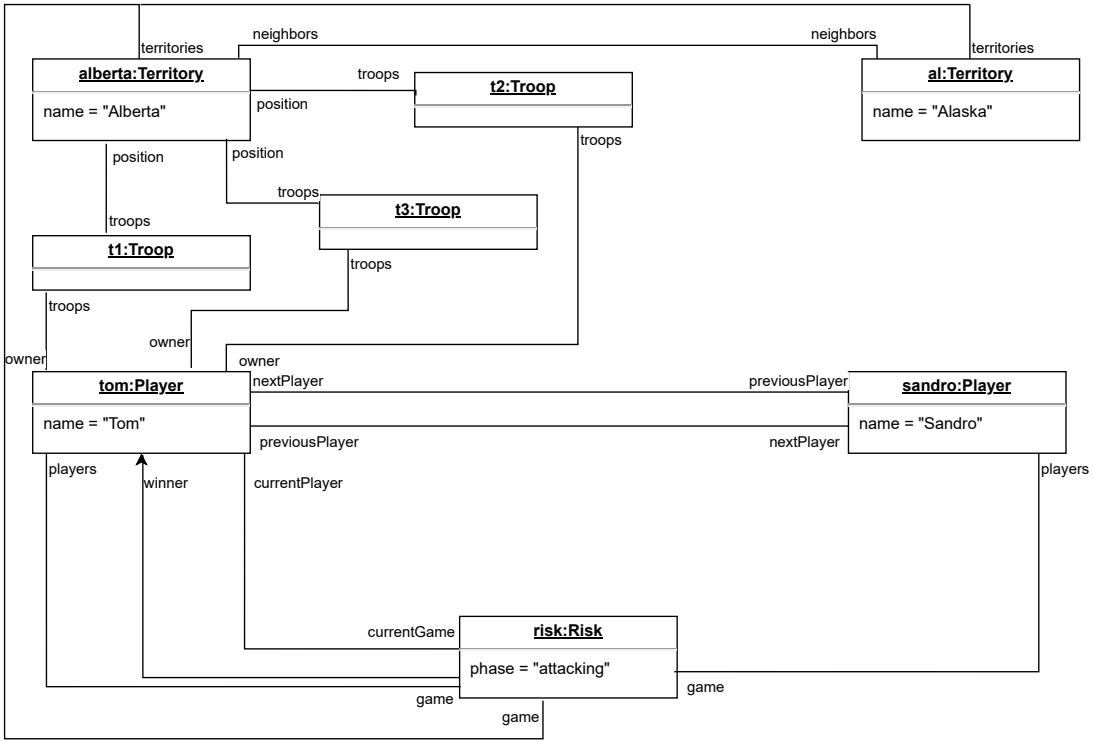

Abbildung 6: Scenario 3: Winning - End

Aufgabe 2 - Testing (11P)

In dieser Aufgabe soll sich grundlegend mit Testing vertraut gemacht werden.

Hierzu ist folgender Code in deinem Repository vorgegeben:

Unter src/main/java im Package de.uniks.ws2324.rps:

- Die Klasse `Player` zur Repräsentation eines Spielers.
- Das Enum `Move` zur Repräsentation von Schere, Stein und Papier als Spielzug.
- Die Klasse `RpsService` für die Logik und zur Steuerung des Schere-Stein-Papier-Spiels.

Unter src/test/java im Package de.uniks.ws2324.rps:

- Die Testklasse `RpsServiceTest`. Nur diese Klasse soll modifiziert werden. Hier schreibst du deine Tests. Was getestet werden soll, wird im Folgenden näher erklärt.

Verwende die vorgegebene Testklasse um folgende Tests zu implementieren:

1. `testFirstPlayerWins`
2. `testIncreasePlayerWins`
3. `testGetBestPlayer`
4. `testPlayRound`

Beachte, dass die Tests mit sinnvollen Asserts durchgeführt werden müssen.

Beim Aufbauen geeigneter Testszenarien darf die Methode `initGame` des `RpsService` genutzt werden.

1. Testmethode: `testFirstPlayerWins`

Hier soll getestet werden, ob die Methode `firstPlayerWins` des `RpsService` entsprechend der Schere-Stein-Papier-Regeln den korrekten Output liefert (für Regeln siehe https://en.wikipedia.org/wiki/Rock_paper_scissors). Hierfür muss kein konkretes Test-Szenario aufgebaut werden. Hinweis: Es sollen alle 9 möglichen Paarungen getestet werden.

2. Testmethode: `testIncreasePlayerWins`

Hier soll getestet werden, ob die Methode `increasePlayerWins` des `RpsService` die Punkte eines Spielers korrekt erhöht.

3. Testmethode: `testGetBestPlayer`

Hier soll getestet werden, ob die Methode `getBestPlayer` des `RpsService` den Spieler mit der höheren Punktzahl zurückgibt.

4. Testmethode: `testPlayRound`

Hier soll getestet werden, ob die Methode `playRound` des `RpsService` zu einer Erhöhung der Rundenzahl führt.

Committe und pushe die Änderungen an deinem Gradle-Projekt abschließend auf den main-Branch.

Bei der Bewertung wird vor allem auf die korrekte Nutzung von JUnit/Assertions geachtet.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Anhang

Es folgt eine Auflistung hilfreicher Webseiten und weiterer Erklärungen zu den Themen dieser Hausaufgabe. Die Links sind als Startpunkt zur selbstständigen Recherche angedacht. Das Durcharbeiten der folgenden Quellen ist kein bewerteter Anteil der Hausaufgaben.

UML

- Was ist UML? https://www.uml.org/what-is-uml.htm
- UML (Spezifikation), nur damit ihr mal gesehen habt, wie so etwas aussieht ;) https://www.omg.org/spec/UML/2.5.1/PDF
- Objektdiagramme: https://mbse.se-rwth.de/book1/index.php?c=chapter4
- Klassendiagramme: https://mbse.se-rwth.de/book1/index.php?c=chapter2

IntelliJ IDEA

Bei IntelliJ wird zwischen der kostenlosen „Community“ und der kostenpflichtigen „Ultimate“-Version unterschieden. Es ist für Studierende möglich die „Ultimate“-Version kostenlos zu erhalten, dies ist die „Free Educational License“. Für diese Veranstaltung genügt die kostenlose Version.

- Download: https://www.jetbrains.com/idea/download/
- Free Educational Licenses: https://www.jetbrains.com/community/education/#students
- Unterschiede der Versionen: https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce