Ultimate Tic-Tac-Toe (20 Punkte + 3 Bonuspunkte)

In diesem Projekt ist es Ihre Aufgabe das Spiel Ultimate Tic-Tac-Toe zu programmieren.

1 Spielprinzip

Ultimate Tic-Tac-Toe ist zusammengesetzt aus 9 Tic-Tac-Toe Feldern, welche in einem 3x3 Feld angeordnet sind. Ein Zug eines Spielers wird auf den kleinen Tic-Tac-Toe Feldern ausgeführt, bis einer der beiden Spieler auf dem größeren Feld gewonnen hat.

1.1 Regeln

- Jedes kleine 3x3 Feld wird im folgenden als *lokales* Spielfeld bezeichnet, das große 3x3 Feld als *globales* Spielfeld.
- Der Spieler mit dem Symbol 'X' startet das Spiel und darf seine Markierung in ein beliebiges der 81 freien Felder setzen.
- In welchem globalen Feld der nächste Spieler seine Markierung setzen darf, ergibt sich aus der Position, die der Gegner in seinem Zug im lokalen Feld gesetzt hat. Als Beispiel siehe Abbildung 1.
- Wenn ein lokales Spielfeld nach den Regeln von normalem Tic-Tac-Toe gewonnen wurde, nimmt dieses Feld im globalen Spielfeld das Symbol des Gewinners an.
- Ein lokales Spielfeld, welches gewonnen wurde oder in welchem jedes Feld markiert wurde, ist *abgeschlossen*. Müsste ein Spieler eine Markierung in ein abgeschlossenes lokales Feld setzen, kann er sich frei entscheiden wohin er die nächste Markierung setzen möchte.
- Das Spiel endet, wenn entweder ein Spieler das globale Feld gewonnen hat oder kein Zug mehr möglich ist, wobei das Spiel dann unentschieden endet.

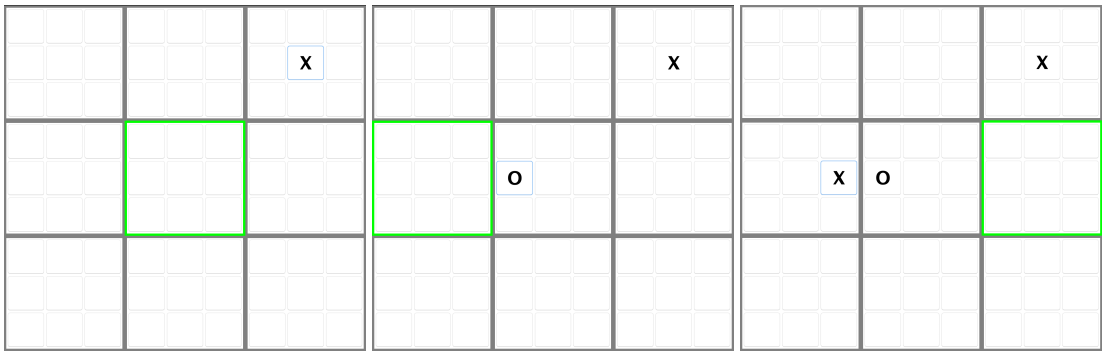

Abbildung 1: Die ersten 3 Spielzüge eines Spiels. Der grüne Rahmen zeigt jeweils in welchem lokalen Feld der nächste Zug stattfinden muss.

2 Aufgabe

Dieses Projekt ist in zwei Phasen unterteilt. In der ersten Phase ist es Ihre Aufgabe Tests zu schreiben, welche Fehler in fehlerhaften Implementierungen finden sollen. In der zweiten Phase sollen Sie dann selbst eine Implementierung des Spiels erstellen.

2.1 Testerstellung (10 Punkte) Woche 1

Beachten Sie, dass zur Bewertung Ihrer Tests der eingebuchte Stand zum Abgabezeitpunkt (1 Woche vor Projektende) herangezogen wird!
In dieser Aufgabe müssen Sie JUnit-Tests implementieren. Diese sollen dazu dienen, Implementierungen der Spezifikation zu überprüfen. In diesem Sinne führen wir Ihre Tests mit sowohl fehlerhaften Implementierungen als auch einer fehlerfreien Implementierung der im Projekt vorhandenen Schnittstellen und Klassen aus. Für jede fehlerhafte Implementierung erwarten wir, dass mindestens einer Ihrer Tests auf dieser fehlschlägt, gleichzeitig aber auf der korrekten Implementierung erfolgreich verläuft.
Beachten Sie, dass es zur Evaluation Ihres Projektes noch mehr fehlerhafte Implementierungen gibt. Schreiben Sie Ihre Tests deshalb gegen die Spezifikation und nicht nur gegen die daily Tests auf dem Server.

Zu testen sind alle Methoden der vorgegebenen Interfaces `MarkInterface`, `BoardInterface` und `SimulatorInterface` außer der Methode **run** aus der Schnittstelle `SimulatorInterface`.
Beachten Sie: Eingaben an Methoden, welche gegen Implementierungsinvarianten verstoßen, sollen zu *IllegalArgumentExceptions* führen. Implementierungsinvarianten sind hier Annahmen über Eingaben an Methoden, die nicht über das graphische Userinterface verletzt werden können. Dazu gehören zum Beispiel, dass ein Eingabezug ein Feld außerhalb des Spielfeldes beschreibt.

Technische Hinweise
Alle Tests, die zur Bewertung dieser Aufgabe herangezogen werden sollen, müssen in dem Java-Paket `uttt.tests` implementiert werden. Es ist unerheblich ob Sie dazu mehrere Klassen benutzen, oder nicht. Sie können bereits einen kleinen Beispieltest in der Klasse `SimpleTest` finden.
Wenn Sie eigene Hilfsmethoden/-klassen für Ihre Tests implementieren möchten, müssen diese ebenfalls in dem `uttt.tests` Paket implementiert sein.
Des Weiteren wird eine von uns gestellte Klasse `UTTTFactory` vorhanden sein. Mit Hilfe dieser können Sie Instanzen der von uns gestellten Implementierungen der Schnittstellen erzeugen.
Außerdem dürfen Sie, da die Testerstellung bewertet wird, hierfür keine Tests von Kommilitonen benutzen.
Als Hilfestellung gibt es den Test `prog2.tests.daily.TestsCorrect.all` der alle Ihre Tests einmal mit der korrekten Implementierung ausführt und die fehlschlagenden Tests ausgibt. Sie können also, wenn Sie sich an einzelnen Stellen der Spezifikation unsicher sind, einfach einen entsprechenden Test schreiben und überprüfen ob Ihr Verständnis der Spezifikation von der Referenzimplementierung bestätigt wird. Nach dem Abgabezeitpunkt werden Ihre Tests weiter durch den daily Test geprüft, aber nicht mehr bewertet.

2.2 Implementierung (10 Punkte) Woche 2

In diesem Teil des Projektes sollen Sie das Spiel, mit Hilfe der schon bekannten Interfaces, selbst implementieren. Ihre Klassen, welche die Interfaces implementieren, müssen Sie im `src.uttt.game` Paket anlegen.
Da Sie die Tests selbst schreiben, wird es zu Ihrer Implementierung keine Publictests geben.

Hinweise: Nutzen Sie die Java-Docs in den Interfaces `MarkInterface`, `BoardInterface` und `SimulatorInterface` als Spezifikation der zu implementierenden Methoden. Im Gegensatz zur Aufgabe in 2.1 müssen Sie alle Methoden implementieren! Sobald Sie Ihre eigene `UTTTFactory` implementieren, können Sie Ihre Tests ganz einfach in Eclipse ausführen, um darüber Ihre eigenen Implementierungen von `ultimate-tic-tac-toe` zu testen.

MarkInterface Das Interface `MarkInterface` beschreibt die Markierungen, welche von den Spielern auf den einzelnen Boards gesetzt werden. Jede Markierung besteht aus dem Symbol des Spielers, der es gesetzt hat sowie der Position in seinem jeweiligen lokalen Board, siehe Abbildung 2 als Beispiel wie die Indices zu wählen sind.

BoardInterface Das Interface `BoardInterface` beschreibt den Aufbau eines einzelnen Tic-Tac-Toe Feldes des Spiels. Auf den `Board`s werden die Markierungen (`Mark`) der einzelnen Spieler gesetzt. Vorher wird überprüft, ob dies ein valider Zug war.

SimulatorInterface Die Implementierung des `SimulatorInterface` soll die Spiellogik bereitstellen, um ein ganzes Spiel ablaufen zu lassen. Dabei wird festgestellt, ob die Spieler auf dem korrekten Board spielen sowie wann ein Spieler gewonnen hat oder das Spiel in einem Unentschieden endet.

PlayerInterface Das `PlayerInterface` gibt die Züge des Spielers an den Simulator weiter. Es kann sowohl die Züge aus der UI, falls ein Mensch spielt, als auch Züge einer KI weiterleiten.

UTTTFactory Die `UTTTFactory` dient dazu, die einzelnen Klassen der Implementierung zu verknüpfen und austauschbar zu gestalten. Hier werden Instanzen der Implementierungen erzeugt und zurück gegeben, um in der Main ein funktionierendes Spiel zu bauen.

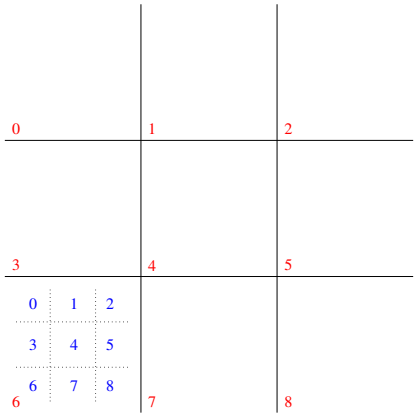

Abbildung 2: Die roten Zahlen stehen für die Board Indices, die blauen für die Mark Indices in einem Board.

Ultimate Tic-Tac-Toe spielen

Nachdem Sie Aufgabe 2.2 implementiert haben, können Sie Ihr selbst implementiertes Spiel spielen. Gehen Sie dazu in Eclipse wie folgt vor: Rechtsklick auf `Main.java` im Paket `uttt.game`, *Run as* → *Java Application*.

2.3 KI (3 Bonuspunkte)

Zum Abschluss des Projekts sollen Sie einen Computerspieler schreiben. Dieser Computerspieler soll das Interface `PlayerInterface` implementieren und auf einem Simulator spielen können, welcher bereits einen Spielstand enthält. Um Bonuspunkte zu erlangen, muss Ihr Computerspieler auf die Spielsituation reagieren und mit möglichst wenigen Zügen gewinnen. Jede Testspielsituation besitzt eine eindeutig kürzeste Lösung die Ihre KI finden muss, um den Testpunkt zu erhalten.

Hinweis: Wenn Sie in ihrer KI zwei Objekte (Simulator, Board oder Mark) mittels der „equals“-Methode miteinander vergleichen müssen Sie diese implementieren, damit es auch lokal funktioniert.

3 Java Projekte — Eclipse

Dieses und die verbleibenden Projekte werden von Ihnen in Java implementiert. Um Ihnen dies zu erleichtern, dürfen Sie die Entwicklungsumgebung *Eclipse*[1][2] benutzen. In der VM ist Eclipse sowie alles weitere Benötigte bereits vorinstalliert.
Falls Sie nicht in der VM arbeiten, benötigen Sie das Java Development Kit (JDK), entweder von Oracle[3] oder die open-source Variante OpenJDK[4]. Auf unseren Servern (und in der VM) ist OpenJDK Version 13 installiert,

[1] `http://de.wikipedia.org/wiki/Eclipse_(IDE)`
[2] `https://github.com/pellaton/eclipse-cheatsheet`
[3] `https://www.oracle.com/java/technologies/javase-jdk13-downloads.html`
[4] `https://jdk.java.net/archive/`

es sollte aber keinen Unterschied machen, wenn Sie stattdessen mit dem Oracle JDK arbeiten. Anhand der daily Tests können Sie vergleichen, ob sich Ihre Implementierung auf unserem Server genauso verhält wie bei Ihnen.
Starten Sie dazu zunächst Eclipse. Wenn Sie nach einem „Workspace“ gefragt werden, übernehmen Sie den Standardpfad (z.B. `/home/prog2/workspace/`) und aktivieren Sie die „Use this as the default ...“ Box bevor Sie mit `OK` bestätigen.
Um das Projekt in Eclipse bearbeiten zu können, müssen Sie erst das Depot auschecken und das Projekt importieren:

1. Klonen Sie das Projekt in einen beliebigen Ordner:
   `git clone https://prog2scm.cdl.uni-saarland.de/git/project3/$NAME /home/prog2/project3`
   wobei Sie $NAME durch ihren CMS-Benutzernamen ersetzen müssen.

2. Benutzen Sie *Import*, ein Unterpunkt des *File* Menüeintrags in Eclipse, um den Importierdialog zu öffnen.

3. Wählen Sie „Existing project into workspace“ aus und benutzen Sie den neuen Dialog um den Ordner des Projekts (siehe Punkt 1) auszuwählen.