Software-Challenge Einstieg

Sven Koschnicke

2018-09-05

Das Spiel kennenlernen

Um ein Computerprogramm zu schreiben, was ein Spiel spielen kann, muss man vorher das Spiel selbst verstehen und spielen können. Die Spielregeln finden sich unter folgendem Link: Piranhas Spielregeln. Das Spiel selbst kann zu zweit oder allein gegen den SimpleClient mit dem Spielleiter gespielt werden. Es empfielt sich, dies einige Male zu tun, bevor man mit der Programmierung beginnt.

Installation von Java

Die meisten Programme, die vom Institut für Informatik zur Verfügung gestellt werden, sind in der Programmiersprache Java geschrieben. Diese Anleitung soll die Beschaffung und Installation von Java erleichtern.

Grundsätzliches

Java gibt es in zwei verschiedenen Paketen: Das *Java Runtime Environment (JRE)* und das *Java Development Kit (JDK)*. Möchte man lediglich Java-Programme starten, also nicht selber entwickeln, dann reicht das JRE vollkommen aus. Möchte man auch eigene Programme schreiben, muss das JDK auf jeden Fall installiert sein. Da im JDK auch das JRE integriert ist, kann man aber immer ohne Bedenken gleich zum JDK greifen.

Installation

Das JDK gibt es auf den Seiten von Oracle http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html#javasejdk Dort das aktuelle "JDK" herunterladen. Es gibt auch Installationsanleitungen auf der Seite.

Installation über Paketquellen (Linux)

Meistens ist das Java JDK in den Paketquellen der Linux-Distributionen enthalten, so dass man es einfach über den Paketmanager installieren kann. Sofern möglich, wird diese Art der Installation empfohlen, da es oft noch Paketabhängigkeiten gibt, die dann automatisch mitinstalliert werden.

Weiterführende Informationen

- Die Java-Seiten von Oracle
- Installation von Java auf Ubuntu Linux (Für andere Distributionen gibt es meist auch Wikis oder Foren mit den entsprechenden Anleitungen)

Einrichtung der (Java-)Entwicklungsumgebung

Die Aufgabe einer Entwicklungsumgebung (IDE) ist es, den Programmierer bei seiner Arbeit zu unterstützen. Dazu bietet sie neben dem Editor auch viele Tools, die das Entwickeln eigener Programme stark erleichtern. Zwei große Vertreter an Entwicklungsumgebungen sind Eclipse und IntelliJ.

Hinweis: Bevor man sich um die Einrichtung der Entwicklungsumgebung kümmert, muss unbedingt Java installiert sein.

SimpleClient beschaffen

Der SimpleClient ist schon ein fertiger Computerspieler. Denn Quellcode kann man verwenden, um seinen eigenen Spieler zu programmieren. Den SimpleClient bekommt man im Downloadbereich der Software-Challenge (https://www.software-challenge.de/downloads). Man braucht die Version "als Quellcode".

Einrichtung von Eclipse

Beschaffung und Installation der Software

Am einfachsten ist die Installation von Eclipse mittels des Eclipse Installer. Dies ist auf folgender Seite erklärt: https://www.eclipse.org/downloads/packages/installer

SimpleClient in Eclipse einbinden

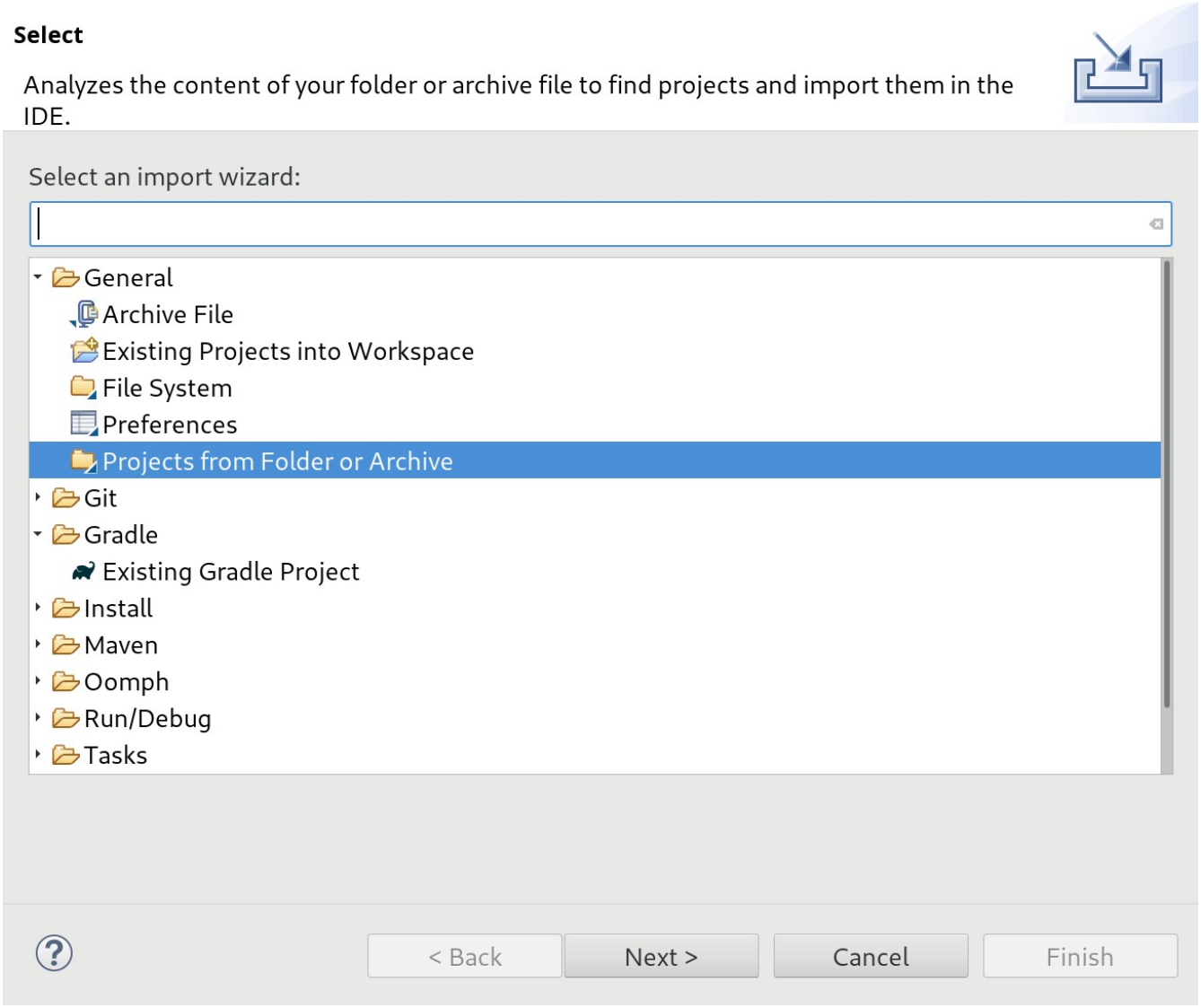


Figure 1. SimpleClient in Eclipse importieren

1. Im Menü auf "File" → "Import..." gehen
2. Im Dialogfenster unter "General" "Projects from Folder or Archive" wählen, dann auf den "Next" Button klicken
3. Oben rechts auf "Archive..." klicken und die heruntergeladene ZIP-Datei mit dem SimpleClient auswählen. Dann auf "Finish" klicken.

Gegebenenfalls ist es notwendig, die mitgelieferte Dokumentation manuell hinzuzufügen. Dazu geht man wie folgt vor:

- Im Package Explorer unter "Referenced Libraries" das Plugin zum aktuellen Spiel suchen. Bei der Software-Challenge 2019 heißt es beispielsweise "piranhas_2019.jar"
- Rechtsklick auf das Plugin und "Properties" anklicken
- Unter "Javadoc Location" lässt sich der "Javadoc location path" angeben. Dort den Ordner "doc" des simple Clients eingeben.

SimpleClient aus Eclipse starten

Den SimpleClient kann man starten, indem man im Project-Explorer einen Rechtsklick auf die Datei

`Starter.java` macht und dann "Run As" → "Java Application" auswählt.

Hinweis: Damit der SimpleClient erfolgreich startet, muss der Spielleiter laufen und auf eine Verbindung warten.

Weiterführende Links

- http://www.eclipse.org Homepage der Eclipse-IDE
- http://www.netbeans.org Homepage des NetBeans-Projektes

Bedienung von Eclipse

Wenn man bisher noch nicht mit einer Entwicklungsumgebung gearbeitet hat, mag der Anblick erschreckend unübersichtlich sein. Sobald man sich jedoch etwas intensiver damit beschäftigt hat, möchte man den Bedienkomfort eines solchen Entwicklertools gar nicht mehr missen. Dieser Artikel stellt die wichtigsten Komponenten der Entwicklungsumgebung Eclipse vor.

Die Oberfläche

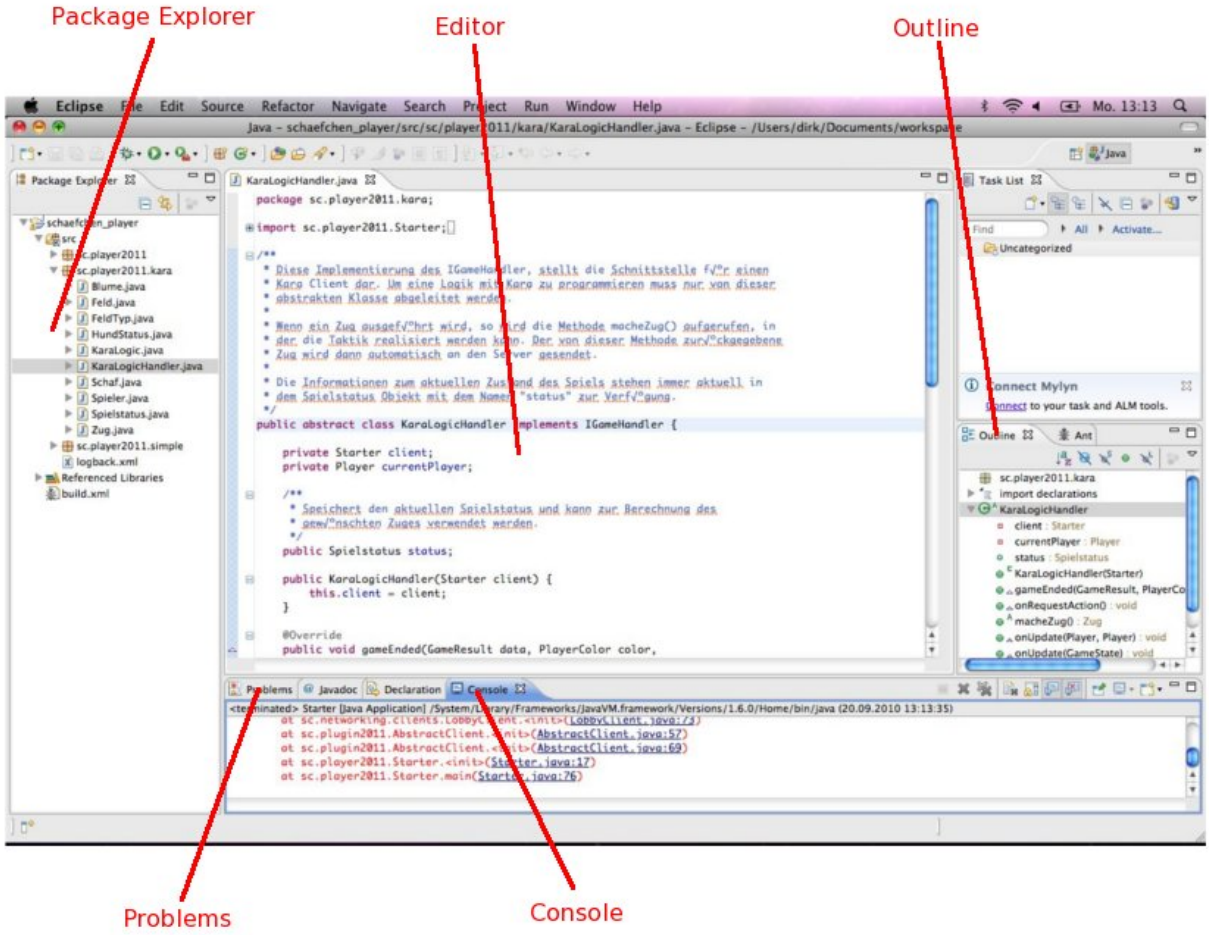


Figure 2. Überblick über die wichtigsten Fenster in Eclipse

Package Explorer

Der Package Explorer befindet sich am linken Rand. Er verwaltet alle importierten Projekte. Wenn man im Package Exlorer einen Doppelklick auf eine Datei macht, wird diese im Editor angezeigt. Mit einem Rechtsklick auf eine Datei oder ein Verzeichnis bekommt man viele Optionen, mit denen sich das ausgewählte Objekt bearbeiten lässt.

Editor

Der Editor ist die große Fläche in der Mitte des Eclipse-Fensters. Am oberen Rand befindet sich die Tab-Leiste, die alle geöffneten Dateien beinhaltet.

Outline

Am rechten Bildschirmrand befindet sich die Outine. Sie zeigt alle Variablen und Methoden der Klasse an, die gerade im Editor geöffnet ist. Mit einem Doppelklick auf einen Eintrag springt der Cursor im Editor an die entsprechende Stelle im Code.

Problems

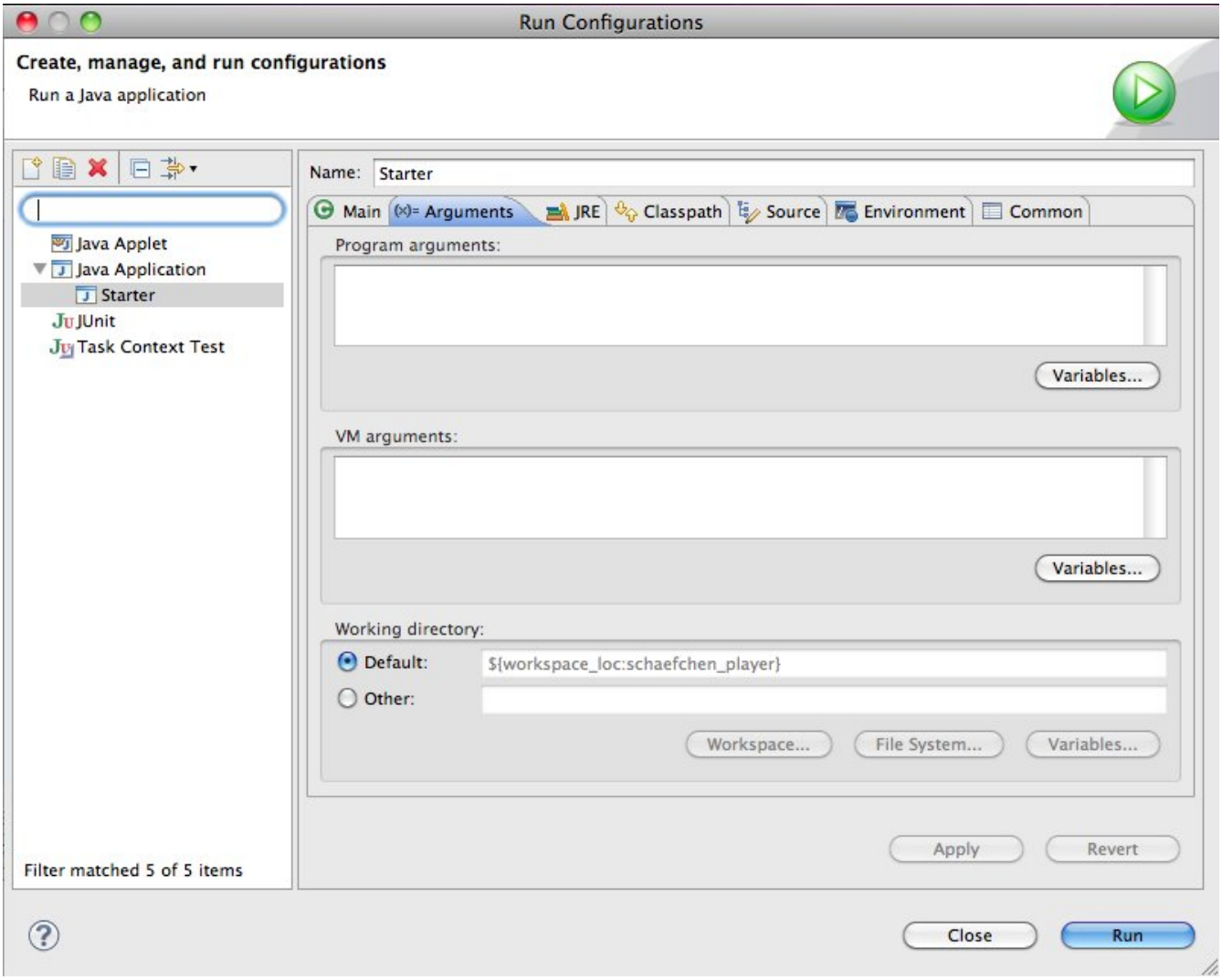
Der Tab Problems befindet sich im Fenster, das am unteren Bildschirmrand zu sehen ist. Hier werden sowohl Programmierfehler, als auch Warnungen angezeigt. Mit einem Doppelklick auf einen Eintrag springt der Cursor im Editor an die entsprechende Codezeile.

Console

Die Console ist nicht sofort sichtbar, sondern erscheint erst, nachdem das erste Programm ausgeführt worden ist. In der Console werden alle Systemausgaben angezeigt. Falls ein Fehler (Exception) geworfen wird, kann man durch einen Klick darauf an die entsprechende Zeile im Programmcode gelangen.

Programme starten

Dialog um die Starteinstellungen des Programms zu ändern



Ein Programm lässt sich starten, indem man im Package Explorer einen Rechtsklick auf die Datei mit der Main-Methode macht und dann "Run As" → "Java Application" ausführt.

Im Menü kann man unter "Run" → "Run Configurations" im Tab "Arguments" noch Optionen angeben.

Tastaturkürzel

Eclipse kennt viele Tastenkombinationen, mit Hilfe derer einige Eclipse-Funktionen schneller aufgerufen werden können. Die wichtigsten Shortcuts kann man der folgenden Tabelle entnehmen:

Aktion	**Effekt**
Strg+Shift+F11	Führt die zuletzt ausgeführte Java-Datei erneut aus
Cursor auf Variablen-, Klassen- oder Methodennamen, dann Alt+Strg+R	Bennennt alle Vorkommen des Namens im ganzen Projekt um
Strg+F1	Wenn man diese Tasenkombination über einen Fehler oder eine Warnung eingibt, kriegt man von Eclipse Verbesserungs-, bzw. Reparaturvorschläge
Strg+I	Rückt den markierten Text sauber ein

Aktion	Effekt
Strg+F7	Kommentiert die markierten Zeilen ein, bzw. aus
Cursor auf Variablen-, Methoden- oder Klassennamen, dann F3	Der Cursor springt zur der Stelle, wo die Variable oder Klasse definiert wurde
Variablen- oder Klassenname teilweise eingegeben, dann Strg+Space	Eclipse liefert Vorschläge zur Vervollständigung
Eingabe von `syso`, dann Strg+Space	Erzeugt `System.out.println`
Eingabe von `for`, dann Strg+Space	Eclipse liefert eine Auswahl an beliebigen `for` -Schleifen
Eingabe von `if`, dann Strg+Space	Liefert eine Auswahl an `if`-Dialogen

Hinweis: Bei Mac OS X wird statt der Strg-Taste meistens die Apple-Taste benutzt.

Die ersten (Programmier-)Schritte

Bekanntlich ist aller Anfang schwer. Deshalb soll hier eine kleine Hilfe gegeben werden, die den Start mit der Entwicklung erleichtern soll.

Einführung in die objektorientierte Programmierung

"Wenn man heute einen Computer kauft, ist er morgen schon veraltet". Kaum ein anderes Gerücht hält sich so stark in der Informatik, wie das über die Kurzlebigkeit. Jedoch gibt es gerade im Bereich der Softwareentwicklung Programmiertechniken und -konzepte, die seit Jahrzehnten nichts von ihrer Aktualität eingebüßt haben. Ein Beispiel dafür ist die Objektorientierung. Sie wurde bereits Mitte der 80er Jahre entwickelt und ist auch heute noch das Grundkonzept moderner Programmiersprachen.

Idee der Objektorientierung

Die Idee der Objektorientierung ist, die Daten und die Funktionen, die auf diese Daten zugreifen, in einer Komponente zu bündeln. Auf die Daten kann man nur über die entsprechenden Funktionen (die man *Methoden* nennt) zugreifen.

Vorgehensweise ohne Objektorientierung

Möchte man zum Beispiel ein Konto ohne Objektorientierung schreiben, so braucht man eine Integer-Variable `int kontostand = 0;`, die Auskunft über den verfügbaren Saldo (z.B. in Cent) gibt. Ein- und Auszahlungen lassen sich durch direkte Wertzuweisungen vornehmen: `kontostand = kontostand + 30;`. Soweit sogut, aber wie werden Auszahlungen gehandhabt? Eine Auszahlung soll nur möglich sein, wenn das Konto ausreichend gedeckt ist:

```
// Beispiel um 40 Geldeinheiten abzuheben
if (kontostand >= 40) {
    kontostand = kontostand - 40;
}
```

Jedes Mal, wenn Geld abgehoben werden soll, muss erst überprüft werden, ob das Konto ausreichend gedeckt ist. Sollte man die if-Abfrage nur an einer Stelle vergessen, läuft man Gefahr, dass der Saldo negativ wird, was nicht möglich sein soll.

Es gibt aber noch weitere Probleme: Durch `kontostand = kontostand + (-25);` kann man quasi durch Einzahlung eines negativen Betrages den Saldo ins Negative treiben. Dies ist in zweifacher Weise kritisch, da zum einen der Kontostand negativ werden kann und es zum anderen keine negativen Einzahlungen geben soll. Also müssen auch Einzahlungen auf ihre Gültigkeit überprüft werden.

Ein weiteres Problem könnte z.B. entstehen, wenn man dem Kontoinhaber einen Dispokredit einräumen will. Dann muss man in jeder if-Abfrage den Dispobetrag hinzufügen. Sollte das an einer Stelle vergessen werden, kann man unter Umständen nichts abheben, obwohl der Dispo noch nicht voll genutzt ist.

Objektorientiert arbeiten

Möchte man ein Konto objektorientiert darstellen, schreibt man zunächst eine *Klasse*, die eine Art Bauplan ist:

```
public class Konto {
    private int kontostand;

    public Konto() {
        kontostand = 0;
    }

    public void einzahlen(int betrag) {
        if (betrag > 0) {
            kontostand = kontostand + betrag;
        }
    }

    public boolean abheben(int betrag) {
        if (kontostand >= betrag) {
            kontostand = kontostand - betrag;
            return true;
        }
        return false;
    }

    public int getKontostand() {
        return kontostand;
    }
}
```

Das Wort `private` vor der Variablen für den Kontostand bedeutet, dass der Zugriff darauf nur innerhalb der Klasse gestattet ist, während `public` Zugriff von überall erlaubt. Möchte man also Geld einzahlen oder abheben, muss man die entsprechenden Methoden nehmen.

Mit dem Schlüsselwort `new` kann man aus dem Bauplan (also der Klasse) ein Objekt erzeugen:

```
Konto konto1 = new Konto(); // Erzeugt ein Konto-Objekt und speichert es in der
Variablen konto1
Konto konto2 = new Konto(); // Erzeugt noch ein Konto und speichert es in einer
anderen Variablen
```

Das `new` führt den Konstruktor aus (`public Konto() { kontostand = 0; }`). Also hat jedes neu erstellte Konto zunächst Kontostand von 0 Geldeinheiten. Ein Konstruktor muss immer den Klassennamen tragen und darf keinen Rückgabewert (nicht einmal `void`) haben. Analog zu Methoden kann man auch einem Konstruktor Argumente übergeben.

Um Geld einzuzahlen oder abzuheben ruft man die Methoden mit dem sog. Punktoperator auf:

```
Konto konto1 = new Konto();            // konto: 0 Geldeinheiten (GE)
Konto konto2 = new Konto();            // konto2: 0 GE

konto1.einzahlen(100);                 // konto1: 100 GE, konto2: 0  GE
konto2.einzahlen(50);                  // konto1: 100 GE, konto2: 50 GE

boolean erfolg = konto1.auszahlen(30); // konto1: 70 GE, konto2: 50 GE, erfolg: true
erfolg = konto2.auszahlen(90);         // konto1: 70 GE, konto2: 50 GE, erfolg: false
```

Die Erweiterung der Klasse um den Dispokredit erweist sich auch als sehr einfach, da nur noch Änderungen in der Klasse notwendig sind:

```
class Konto {
     private int kontostand;
     private int dispo; ①

     public Konto() {
         kontostand = 0;
         dispo = 500; ②
     }

     public void einzahlen(int betrag) {
         if (betrag > 0) {
             kontostand = kontostand + betrag;
         }
     }

     public boolean abheben(int betrag) {
         if ((kontostand + dispo) >= betrag) { ③
             kontostand = kontostand - betrag;
             return true;
         }
         return false;
     }

     public int getKontostand() {
         return kontostand;
     }
}
```

① Neue private Variable um den Kreditrahmen zu speichern.

② Kreditrahmen im Konstruktor initialisieren.

③ Kreditrahmen beim Abheben mit berücksichtigen.

Referenzierung von Objekten

Wenn man mit `Konto konto3 = new Konto()` ein neues Objekt erzeugt, wird dieses im Arbeitsspeicher abgelegt und die Variable `konto3` enthält die Speicheradresse zum entsprechenden

Objekt. Mit dem Befehl `Konto konto4 = konto3` wird der Variablen `konto4` die Speicheradresse von `konto3` zugewiesen. Beide zeigen also auf dieselbe Speicheradresse und somit auf das gleiche Objekt. Somit verändert `konto3.einzahlen(40)` auch den Kontostand von `konto4`, weil beide auf dasselbe Objekt zeigen. Statt *zeigen* sagt man oft auch *referenzieren*.

Merkregel: Neue Objekte erzeugt man nur mit dem Schlüsselwort `new`!

Vererbung

Die Vererbung ist eine Technik, mit der man eine Klasse, durch hinzufügen von Methoden und Variablen, einen neuen Bauplan (Klasse) erzeugt.

Möchte man zum Beispiel zusätzlich auch noch ein Premiumkonto anbieten, auf dem der Kontostand verzinst wird, kann man die bestehende Klasse nehmen und entsprechend erweitern:

```
public class PremiumKonto extends Konto {
     private double zinsbetrag;

     public PremiumKonto() {
         super();
         zinsbetrag = 2.5d; // 2.5% Zinsen
     }

     public void zinsenGutschreiben() {
         int saldo = getKontostand();
         if (saldo > 0) {
             einzahlen(saldo * zinsbetrag / 100);
         }
     }
}
```

Die Methoden zum Ein- und Auszahlen brauchen nicht neu geschrieben werden, da diese von der Klasse Konto "kopiert" werden. Man kann eine Methode aus einer Oberklasse neu schreiben. Dann wird immer die geänderte Version genommen. Das Schlüsselwort `super()` ruft den Konstruktor aus der Kontoklasse auf. In Java wird immer der leere Konstruktor der Oberklasse aufgerufen, so dass diese Zeile auch weggelassen werden darf.

Ein neues Objekt erzeugt man auf die gleiche Weise, wie bei einem normalen Konto:

```
PremiumKonto premium = new PremiumKonto();

premium.einzahlen(50); //geerbte Methode
premium.zinsenGutschreiben();
```

Casting von Objekten

Da ein Premiumkonto auch ein normales Konto ist, ist der folgende Aufruf legal:

`Konto konto5 = new PremiumKonto();`

Weil `konto5` vom Typ `Konto` ist, dürfen auch nur die Methoden aus dieser Klasse verwendet werden. Möchte man auch Zinsen gutschreiben können, so muss aus dem Konto ein Premiumkonto gemacht werden:

`PremiumKonto konto6 = (PremiumKonto) konto5;`

Dieser Cast gelingt jedoch nur, wenn das Konto auch ein Premiumkonto ist! Sonst wird eine Fehlermeldung geworfen. Mit dem Schlüsselwort `instanceof` kann man abfragen, ob ein Objekt zu einer gewissen Klasse gehört:

```
Konto konto7 = new PremiumKonto();

if (konto6 instanceof PremiumKonto) {
    Premiumkonto premium2 = (PremiumKonto) konto7;
    premium2.zinsenGutschreiben();
}
```

Wichtig: Es werden nur Methoden vererbt, jedoch keine Variablen! Deshalb wird auf den kontostand nur über die entsprechenden Methoden der Oberklasse zugegriffen.

Statische Variablen und Methoden

Gibt es Methoden oder Variablen, die für alle Objekte gültig sind, so werden diese als statisch (`static`) deklariert. Statische Variablen und Klassen werden von allen Objekten geteilt.

Soll zum Beispiel der Zinssatz beim Premiumkonto für alle Konten gleich sein, kann man diesen als statisch deklarieren:

```
public class PremiumKonto extends Konto {
     private static double zinsbetrag = 2.5d; // 2.5% Zinsen

     ...

     public static double getZinsbetrag() {
         return zinsbetrag;
     }

     public static void setZinsbetrag(double wert) {
         zinsbetrag = wert;
     }
 }
```

Von außen kommt man an den Zinsbetrag über die Methode `setZinsbetrag(double wert)`, die man entweder über das Objekt oder über den Klassennamen aufrufen darf.

```
PremiumKonto.setZinsbetrag(3d); // Zinsen auf 3% erhöhen

PremiumKonto premium3 = new PremiumKonto();
premium3.setZinsbetrag(3d);
```

Tipp: Damit man besser erkennen kann, dass es sich um statische Variablen oder Methoden handelt, sollte man auf diese immer über den Klassennamen zugreifen.

Weitere Aspekte

Die Objektorientierung bietet noch viele weitere Aspekte, wie zum Beispiel die Polymorphie. Da es sich hier nur um eine Einführung handelt, wurden solche fortgeschrittenen Themen allerdings nicht behandelt.

Weiterführende Informationen

- Eintrag aus der Wikipedia
- Praxisbuch Objektorientierung (openbook)

Der Computerspieler (Client)

Der Computerspieler ist ein Programm, dass sich mit dem Spielleiter (siehe Der Spielleiter (Server)) verbindet und die gestellte Aufgabe selbständig lösen kann. Die Aufgabe der Schüler ist es, sich eine Strategie zu überlegen und zu implementieren, mit der sie gegen die Clients der anderen Schulen gewinnen können.

Der Computerspieler kann in einer beliebigen Programmiersprache geschrieben sein, jedoch gibt es Muster-Computerspieler nur in Java und Ruby.

Die Muster-Computerspieler können im Downloadbereich der Software Challenge Website (http://www.software-challenge.de) heruntergeladen werden.

Hinweis: Das Spielleiter-Programm (siehe Der Spielleiter (Server)) benötigt Java. Deshalb muss auf den ausführenden Rechnern auch das Java SDK installiert sein.

Der SimpleClient

Der SimpleClient ist ein Computerspieler, den das Institut für Informatik ins Rennen schickt. Er stellt zwar eine korrekte Lösung der gestellten Aufgabe dar, ist aber nicht besonders intelligent. Neben dem eigentlichen Programm ist auch der Quellcode des SimpleClients verfügbar. Auf diese Weise können sich die Schüler anschauen und lernen, wie man die gestellte Aufgabe lösen kann. Außerdem darf der Code um die eigene Strategie erweitert werden. Auf diese Weise müssen die Schüler nicht den ganzen Computerspieler selbst entwickeln, sondern können sich auf den Entwurf und die Implementierung ihrer eigenen Strategie konzentrieren.

Der NotSoSimpleClient

Wenn die aktuelle Saison der Software-Challenge etwas weiter fortgeschritten ist, stellt das Institut einen stärkeren Computerspieler zur Verfügung: den NotSoSimpleClient. Das ist ein Spieler, der eine effizientere Strategie zur Lösung der Aufgabe als der SimpleClient verfolgt und dadurch nicht mehr so leicht zu schlagen ist. Dieser Spieler wird ohne den Quellcode veröffentlicht, so dass die Schüler den NotSoSimpleClient zwar als Gegenspieler für Testspiele nehmen, jedoch nicht den Quellcode für den eigenen Spieler weiterverwenden können.

Der Spielleiter (Server)

Die beiden Computerspieler kommunizieren nicht direkt miteinander, sondern übertragen ihre Nachrichten über einen Mittelmann: den Spielleiter. Dadurch ist zum einen sichergestellt, dass man seinen Gegner nicht mit illegalen Nachrichten belästigen kann, zum anderen sorgt der Spielleiter dafür, dass sich die Kontrahenten an die Spielregeln halten.

Der Spielleiter ist direkt im Wettkampfsystem integriert, so dass alle Turnierspiele regelkonform gespielt werden müssen. Zum Testen des eigenen Computerspielers gibt es eine spezielle Version des Spielleiters, die im Downloadbereich (http://www.software-challenge.de) heruntergeladen werden kann. Diese Download-Version, die in diesem Abschnitt beschrieben wird, besitzt eine grafische Oberfläche, durch die man das Spiel gut verfolgen und sogar als Mensch mitspielen kann.

System vorbereiten und Spielleiter starten

Die einzige Vorraussetzung ist, dass auf dem Rechner mindestens die Laufzeitumgebung für Java 8 installiert ist. Siehe Installation von Java.

Nach der erfolgreichen Installation kann man den Server durch einen Doppelklick auf die Datei `software-challenge-gui` starten.

Die Programmoberfläche

Die Programmoberfläche besteht aus dem Menü links sowie der Spielfläche in der Mitte.

Figure 3. Die Spielleiteroberfläche direkt nach dem Programmstart

Auf der linken Seite sind alle möglichen Aktionen aufgelistet.

Zu Beginn ist die Spielfläche leer.

Ein neues Spiel erstellen

Um ein Spiel zu spielen, muss zunächst über das Menü links "Neues Spiel" die folgende Ansicht aufgerufen werden:

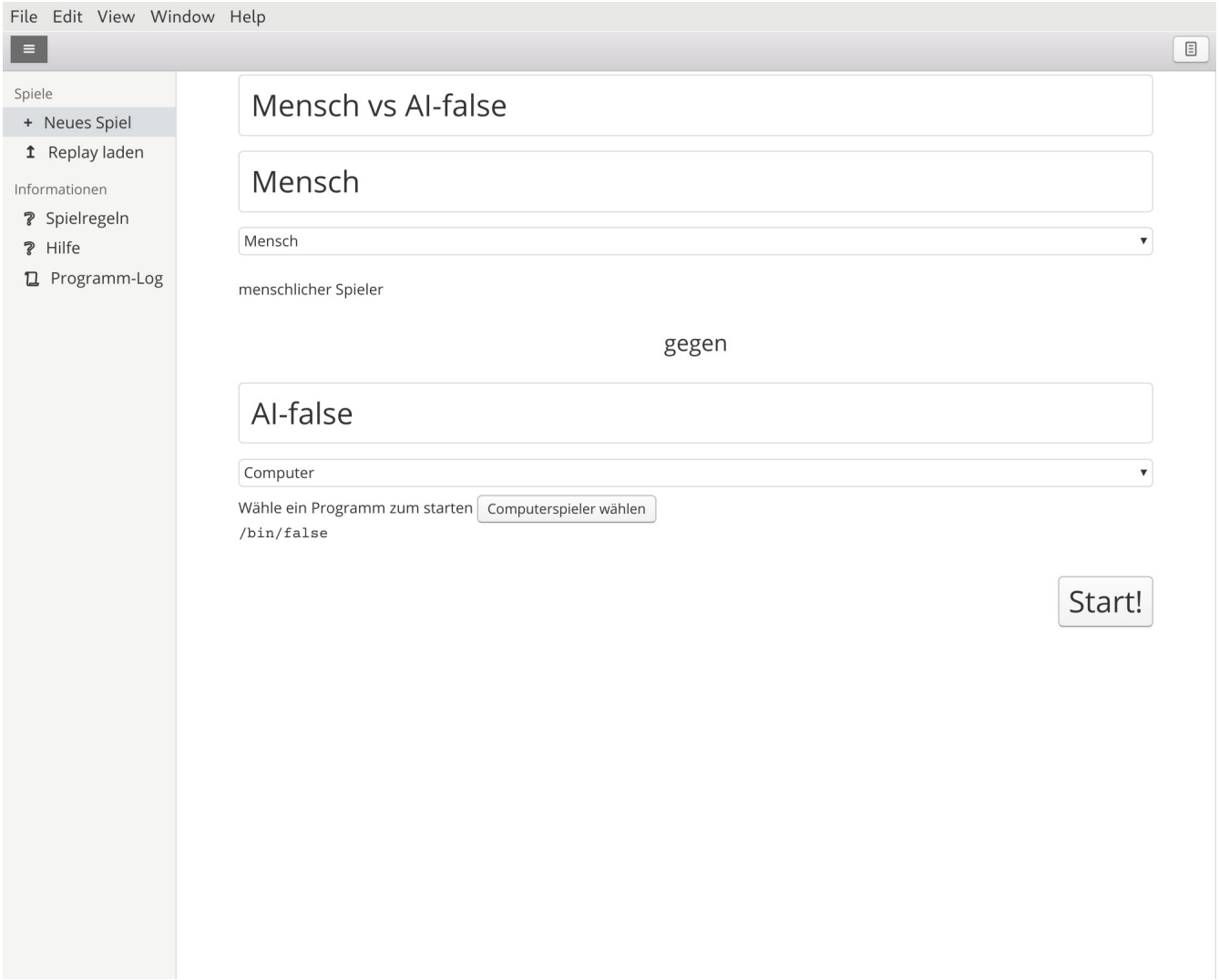


Figure 4. Dialog für ein neues Spiel

In diesem Fenster werden die Spieler ausgewählt, die an dem Spiel teilnehmen sollen. Für jeden Spieler gibt es folgende Optionen:

- Name: Hier kann für jeden Spieler ein Name eingegeben werden, der dann im Spiel angezeigt wird.
- Spielertyp: Es kann zwischen 3 verschiedenen Spielertypen gewählt werden:
  1. Mensch: Ein menschlicher Spieler, der über die Programmoberfläche spielt.
  2. Computerspieler: Ein Computerspieler in Form eines separaten Programms, das beim Starten des Spiels automatisch vom Server gestartet wird.
  3. Manuell gestarteter Client: Ein Computerspieler in Form eines separaten Programms, das manuell durch den Benutzer gestartet werden muss.
- Wähle ein Programm zum starten (auswählbar für 'Computerspieler'): Bei Spielertyp ‚Computerspieler' ist eine Programmdatei auszuwählen, deren Name dann hier angezeigt wird.

Nach Eingabe der erforderlichen Werte kann das Spiel mithilfe des unteren Knopfs "Start!" erstellt werden.

Die Spielfeldoberfläche

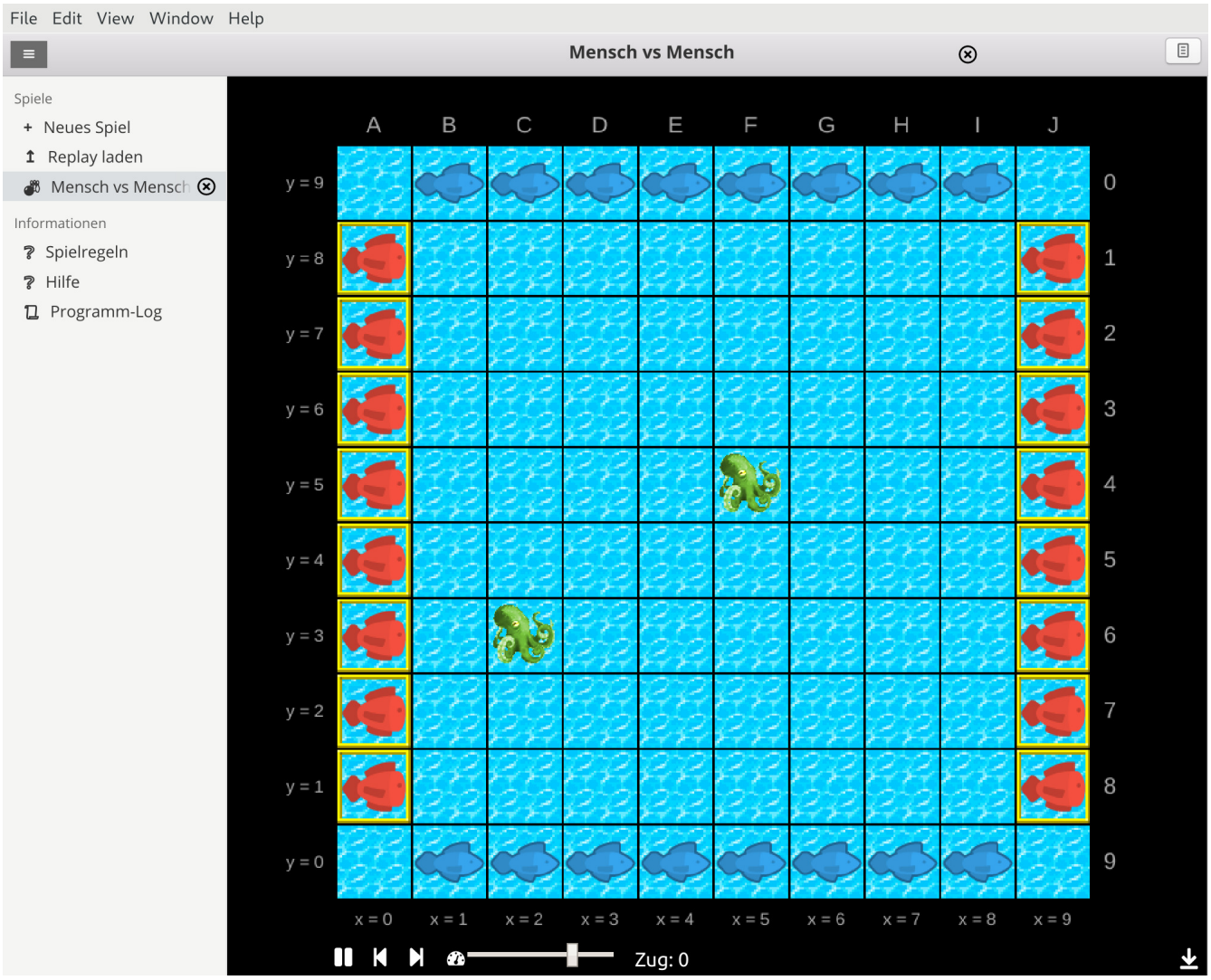


Figure 5. Die Spielfeldoberfläche (hier mit dem Spiel "Piranhas")

Die Spielfeldoberfläche setzt sich aus dem Spielbrett (der große zentrale Bereich) und den Spielsteuerelementen (unten) zusammen.

Auf dem Spielbrett werden das eigentliche Spiel (hier z.B. "Piranhas"), die Züge und weitere für das Spiel wichtige Informationen dargestellt. Hier setzt der menschliche Spieler auch seine Züge.

Die Steuerelemente unterscheiden sich je nach Spiel und Spielsituation. Unten links gibt es immer die Schaltflächen "Abspielen/Pause", "Vor" und "Zurück". Mit diesen kann man die Züge des Spiels durchgehen (das ist am nützlichsten, wenn zwei Computerspieler gegeneinander spielen und man sich die Züge genau ansehen will). Rechts daneben gibt es noch einen Regler für die Abspielgeschwindigkeit und eine Anzeige des aktuellen Zuges. Ganz rechts ist die Schaltfläche zum Speichern des aktuellen Spiels als Replay-Datei, die man dann später wieder laden und das Spiel erneut ansehen kann.

Wenn ein menschlicher Spieler am Zug ist, werden ausserdem Schaltfächen zur Eingabe des Spielzuges angezeigt. Diese sind von Spiel zu Spiel sehr unterschiedlich.

Für das Spiel Piranhas:

- Die Fische, die man ziehen kann, werden mit einem gelben Kasten markiert. Um einen Fisch zum ziehen auszuwählen, klickt man auf ihn. Der ausgewählte Fisch beginnt dann zu springen.
- Nach Auswahl eines zu ziehenden Fisches werden die möglichen Zielfelder mit einem gelben Kasten markiert. Um den Fisch auf ein mögliches Feld zu ziehen, klickt man es an. Der Fisch zieht auf das angeklickte Feld und der Zug ist beendet.
- Möchte man nach Auswahl eines Fisches doch einen anderen Fisch ziehen, klickt man irgendwo auf dem Spielbrett, nur nicht auf eines der markierten Zielfelder. Dann wird der Fisch wieder abgewählt und man kann erneut einen zu ziehenden Fisch auswählen.

Spielwiederholungen

Spielwiederholungen oder Replay-Dateien sind aufgezeichnete frühere Spiele, die man sich beliebig oft wieder ansehen kann, um beispielsweise einen Fehler des eigenen Spielers zu analysieren oder eine Strategie zu verbessern.

Um das aktuelle Spiel als Spielwiederholung zu speichern, klickt man auf das Icon ganz rechts unten im Spielbereich. Dann kann man einen Dateinamen und Speicherort festlegen.

Um eine gespeicherte Spielwiederholung zu laden verwendet man den Eintrag "Replay laden" in der linken Leiste. Nachdem man eine Datei ausgewählt hat, kann man das gespeicherte Spiel abspielen oder Schritt für Schritt durchgehen.

Testdurchläufe

Wenn man einen grundsätzlich funkionierenden Computerspieler programmiert hat, ist es sinnvoll, diesen mit vielen verschiedenen Spielsituationen zu konfrontieren. Dadurch lassen sich Fehler entdecken und die Spielstärke des Computerspielers beurteilen. Für solche Testdurchläufe wird ein Testserver und TestClient zur Verfügung gestellt. Wie man diese verwendet, ist weiter unten unter den Überschrift Automatische Spiele: Der Testserver und Massentests beschrieben.

Spielsituation nachstellen

Wenn Sie ein Fehlerverhalten Ihres Computerspielers beobachtet haben, das nur in einer bestimmten Situation in einem Spiel aufgetreten ist, kann es oft wünschenswert sein, diese Situation erneut nachspielen zu können, um den Computerspieler gezielt zu verbessern.

Dies ist zur Zeit nur auf etwas kompliziertem Wege möglich. Es folgt eine Schritt-für-Schritt Anleitung:

1. Laden Sie das betreffende Replay aus dem Wettkampfsystem herunter (.xml.gz Datei).
2. Entpacken Sie das Replay, sodass sie eine .xml-Datei erhalten.
3. Starten Sie den Server und erstellen Sie ein neues Spiel. Wählen Sie den Computerspieler, der für diese Spielsituation getestet werden soll. Dieser Spieler muss als Spieler 1 gestartet werden und ist dann direkt als erstes dran. Der Gegenspieler kann dann ein beliebiger Computerspieler oder auch ein Mensch sein.
4. Setzen Sie einen Haken bei "Spiel aus Datei laden". Wählen Sie über "Datei wählen" das

entsprechende Replay aus und spezifizieren sie den Zug in dem gestartet werden soll. Starten Sie dann das Spiel. Das Spiel sollte sich nun in genau der Situation befinden, in der das Fehlerverhalten aufgetreten ist. Dabei ist der Spieler, der nun dran ist immer der rote Spieler. Falls der blaue Spieler eigentlich dran war, werden die Farben der Spieler getauscht.

5. Nun kann der nächste Zug beim Spieler angefordert werden und dabei durch Debugging kontrolliert werden, wo sich der Spieler falsch verhalten hat. Achtung: Wenn weitere Züge angefordert werden, kann das Verhalten vom normalen Spielverlauf abweichen, da evtl. nicht alle Daten für das Spiel in der XML vorhanden sind.

Replay mit Server ohne graphische Oberfläche speichern

Wenn der Server ohne die graphische Oberfläche gestartet wird, kann das `--saveReplay` Attribut gesetzt werden, damit bei Ende jedes Spiels das Replay des Spiels unter `./replays` gespeichert wird.

```
  java -Dfile.encoding=UTF-8 -Dlogback.configurationFile=logback.xml -jar
softwarechallenge-server.jar --port 13051 --saveReplay true
```

Automatische Spiele: Der Testserver

Wenn Sie automatisiert Spiele mit Ihrem Computerspieler spielen wollen, um bestimmte Verhaltensweisen bei der Weiterentwicklung regelmäßig zu testen, können Sie dafür einen speziellen Server ohne grafische Oberfläche verwenden, den sogenannten Testserver. Hier ist zu beachten, dass der Testserver auf dem Port 13051 gestartet wird und nicht, wie im normalen Spiel auf Port 13050.

Gehen Sie dazu wie folgt vor:

1. Laden Sie den Testserver von der Download-Seite herunter.
2. Entpacken Sie das heruntergeladene Archiv.
3. Wechseln Sie in einer Kommandozeilenumgebung (Windows: cmd.exe oder Powershell, Linux: beliebige Shell oder Terminal) in das Verzeichnis des entpackten Archives.
4. Starten Sie den Testserver auf dem Port 13051 mit folgendem Befehl:

   ```
     java -Dfile.encoding=UTF-8 -Dlogback.configurationFile=logback.xml -jar
   softwarechallenge-server.jar --port 13051
   ```

5. Starten Sie Ihren Computerspieler und einen zweiten Computerspieler manuell auf dem Port 13051 (im SimpleClient geht dies mit der Option --port 13051) in weiteren Kommandozeilenumgebungen. Die Computerspieler verbinden sich automatisch zum Testserver und es wird ein Spiel gespielt. Danach sollten sich die Computerspieler automatisch beenden.
6. Wenn Sie weitere Testspiele starten wollen, können Sie die Computerspieler erneut starten. Der

Testserver muss nicht neu gestartet werden.

Beachten Sie, dass der Testserver keine Spielaufzeichnungen anlegt, wie es der Server mit grafischer Oberfläche tut. Die Auswertung der Spiele muss in einem der teilnehmenden Computerspieler geschehen (z.B. durch Log-Ausgaben).

Es ist ebenfalls möglich, statt eines Zufällig generierten vollständigen Spielplanes eine Spielsituation zu laden und zu testen. Die Spielsituation muss vorher wie unter Spielsituation nachstellen erzeugt werden. Dann kann die Datei mit dem Argument `--loadGameFile` geladen werden und optional mit `--turn` ein Zug spezifiziert werden.

```
  java -Dfile.encoding=UTF-8 -Dlogback.configurationFile=logback.xml -jar
softwarechallenge-server.jar --port 13051 --loadGameFile ./replay.xml --turn 10
```

Unerwartete Zugzeitüberschreitungen (Soft-Timeout)

Wenn Sie den Testserver einige Zeit laufen lassen, um eine größere Anzahl von Testspielen durchzuführen, kann es dazu kommen, dass Computerspieler wegen Zugzeitüberschreitungen vom Server disqualifiziert werden (Soft-Timeout). Dies passiert, obwohl sie ihren Zug innerhalb der erlaubten Zugzeit (abhängig vom Spiel, bisher aber immer zwei Sekunden) an den Server geschickt haben. Der Garbage Collector der Java Virtual Machine löst dieses Verhalten aus. Er pausiert die Anwendung, um nicht mehr genutzten Speicher freizugeben. Wenn der Server dadurch zu einem ungünstigen Zeitpunkt angehalten wird, bemerkt er den Eingang des Zuges vom Computerspieler nicht rechtzeitig und disqualifiziert ihn daraufhin. Damit dieses Problem möglichst selten auftritt, haben sich die folgenden Parameter beim Starten des Servers bewährt:

Unter Linux:

```
java -Dfile.encoding=UTF-8 \
     -Dlogback.configurationFile=logback.xml \
     -server \
     -XX:MaxGCPauseMillis=100 \
     -XX:GCPauseIntervalMillis=2050 \
     -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled \
     -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 \
     -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark \
     -jar softwarechallenge-server.jar --port 13051
```

Unter Windows (unterscheidet sich nur durch die Art, den langen Befehl auf mehrere Zeilen zu verteilen):

```
java -Dfile.encoding=UTF-8 ^
     -Dlogback.configurationFile=logback.xml ^
     -server ^
     -XX:MaxGCPauseMillis=100 ^
     -XX:GCPauseIntervalMillis=2050 ^
     -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled ^
     -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 ^
     -XX:+ScavengeBeforeFullGC -XX:+CMSScavengeBeforeRemark ^
     -jar softwarechallenge-server.jar --port 13051
```

Um das Verhalten des Garbage Collectors noch weiter zu verbessern, kann man auch noch mittels der Optionen

```
-XX:+PrintGCDateStamps -XX:+PrintGC -XX:+PrintGCDetails -Xloggc:"pfad_zum_gc.log"
```

eine Logdatei über die Aktivitäten des Garbage Collectors anlegen. Darin sieht man genau, wann er wie lange lief. Man kann dann die Einstellungen verändern und testen, ob sich das Verhalten verbessert.

Die Konfiguration des Garbage Collectors ist kein Allheilmittel und kann zu neuen Problemen führen, auf die man gefasst sein sollte. Dazu gehören erhöhter Resourcenverbrauch und Instabilität der Anwendung.

Massentests mit Server ohne graphische Oberfläche

Wenn Sie Massentests mit ihrem Computerspieler ausführen wollen, um Beispielsweise seine Gewinnchance gegenüber einer früheren Version zu testen, wobei sich die beiden Spieler als Startspieler abwechseln, dann ist dies mit folgenden Schritten möglich:

Starten sie den Server auf dem Port 13051.

```
java -Dfile.encoding=UTF-8 -Dlogback.configurationFile=logback.xml -jar
softwarechallenge-server.jar --port 13051
```

Variante mit TestClient

Starten sie (mit weiterhin laufendem Server, s.o.) den TestClient

```
java -jar -Dlogback.configurationFile=logback-tests.xml test_client.jar
    --tests 4
    --name1 "displayName1"
    --player1 "./player1.jar"
    --timeout1 true
    --name2 "displayName2"
    --player2 "./player2.jar"
    --timeout2 true
    --port 13051
```

Argumente des TestClients

Attribut	Standardwert	Kurzbeschreibung
--tests	100	Anzahl der Tests, die gespielt werden sollen (Integer)
--player1	"./defaultplayer.jar"	Erster Computerspieler (Dateipfad)
--player2	"./defaultplayer.jar"	Zweiter Computerspieler (Dateipfad)
--name1	"player1"	Name des ersten Spielers (String)
--name2	"player2"	Name des zweiten Spielers (String)
--timeout1	true	Scheidet der erste Spieler bei einem Timeout aus? (true|false)
--timeout2	true	Scheidet der zweite Spieler bei einem Timeout aus? (true|false)
--port	13051	Der Port, auf dem der TestClient die Clients startet

Bei Argumenten, die nicht angegeben wurden, werden die Standardwerte aus der Tabelle verwendet. Die Ausgabe der Daten erfolgt nach jedem Spiel anhand von gerundeten Werten. Der TestClient beendet sich selbst, nachdem alle Spiele gespielt wurden.

Die Ergebnisse der Spiele werden für den jeweiligen Spielernamen vom Server zusammengezählt. Dies wird auch über mehrere Starts des TestClients getan. Die Ergebnisse werden erst zurückgesetzt, wenn der Server neu gestartet wird. Achten Sie also darauf, den Server neuzustarten oder einen anderen Spielernamen zu verwenden, wenn Sie den Spieler verändern.

Variante ohne TestClient

Auch hier muss der Server laufen (s.o.).

Starten Sie ein Spiel mit Reservierungscode (siehe Spielverlauf in der XML-Dokumentation). Aktivieren Sie mit dem erstellten Administratorclient den Testmodus:

```
<testModus testModus="true"/>
```

Dies liefert die Antwort

```
<testing testModus="true"/>
```

Mit false als entsprechenden Parameter kann dieser wieder deaktiviert werden. Nun können sie jederzeit die Testdaten der Spieler anhand ihres Anzeigenamens erfragen (es ist zu beachten, dass dafür die Spieler unterschiedliche Anzeigenamen haben müssen):

```
<scoreForPlayer displayName="player1" />
```

Der Server antwortet mit:

```
<playerScore>
  <score displayName="player1" numberOfTests="4">
    <values>
      <fragment name="Gewinner">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <value>4</value>
    </values>
    <values>
      <fragment name="Ø Feldnummer">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <value>5.0000013</value>
    </values>
    <values>
      <fragment name="Ø Karotten">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <value>40.500011</value>
    </values>
  </score>
</playerScore>
```

Bei dieser Variante muss sich selbst um das Starten der Clients gekümmert werden.

Den SimpleClient erweitern

In der Version des Java SimpleClients von der Software Challenge Homepage ist bereits eine

Strategie implementiert, die RandomLogic. Man kann jedoch auch noch beliebig viele eigene Strategien hinzufügen.

Erstellen einer neuen Strategie

Die einfachste Möglichkeit ist, die Klasse `Logic` des SimpleClient zu kopieren und umzubenennen (alle Vorkommen von `Logic` durch den neuen Klassennamen ersetzen). Der Vollständigkeit halber hier noch das Vorgehen bei einer komplett neuen Klasse:

- Erstellt eine neue Klasse (z.B. `MyLogic`), die das Interface `IGameHandler` implementiert:

```
public class MyLogic implements IGameHandler {
    private Starter client;
    private GameState gameState;
    private Player currentPlayer;
```

- Erstellt einen Konstruktor, der eine Instanz des Starters erhält. Diese wird später noch gebraucht

```
public MyLogic(Starter client) {
    this.client = client;
}
```

- Implementiert die 5 Interface-Methoden

```
@Override
public void gameEnded(GameResult result, PlayerColor color, String errorMessage) {
    // Hier muss nichts getan werden
}

@Override
public void onUpdate(Player player, Player otherPlayer) {
    // Der Spieler wurde aktualisiert
    this.player = player;
}

@Override
public void onUpdate(GameState gameState) {
    // Ein neuer Spielstatus, d.h. etwas ist geschehen. Deshalb
    // alles aktualisieren.
    this.gameState = gameState;
    this.player = gameState.getCurrentPlayer();
}

@Override
public void sendAction(Move move) {
    // Einen Zug an den Server senden
    starter.sendMove(move);
}

@Override
public void onRequestAction() {
    // Ich soll einen Zug machen
    Move move;
    // ... Hier muss die Logik rein, die einen Zug findet.
    sendAction(move);
}
```

Nun kann die Strategie in der Methode `onRequestAction` (oder in eigenen Klassen, die dort verwendet werden) implementiert werden.

Eine Idee implementieren

Man hat einige Spiele absolviert und sich eine gute Strategie ausgedacht. Damit hat man zwar schon einen wichtigen Teil der Arbeit geleistet, aber irgendwie muss dem Computerspieler noch beigebracht werden, nach dieser Strategie zu spielen.

Anhand einer kleinen Aufgabe soll gezeigt werden, wie man eine Idee formal beschreiben und in ein Programm überführen kann. Dabei nehmen wir an, dass wir einen Stapel mit Karten haben, der sortiert werden soll.

Vorraussetzungen

- eine beliebige Anzahl an Spielkarten
- eine Reihenfolge, in der die Spielkarten sortiert werden sollen

Idee formalisieren

Als erstes muss die Idee formal beschrieben werden. Oftmals kann man zunächst beschreiben, wie man als Mensch vorgehen würde.

1. Gehe den Stapel durch und merke die Position, an der sich die kleinste Karte befindet.
2. Tausche die Position der kleinsten Karte mit der untersten Karte im Stapel.
3. Die kleinste Karte ist jetzt an der richtigen Position.
4. Führe die Schritte nochmal für den Reststapel (ohne die sortierten Karten) aus.

Idee implementieren

Nachdem man seine Idee formal niedergeschrieben hat, kann sie ganz leicht in ein Programm überführt werden:

```java
/**
 * Das Array a[] symbolisiert den Stapel der unsortierten Karten. Dabei steht
 * eine Zahl immer für eine spezielle Karte. Eine kleinere Zahl bedeutet,
 * dass es sich um eine kleinere Karte handelt.
 *
 * start gibt die Position an, wo der Reststapel beginnt (am Anfang: start = 0)
 */
public static void sortiere(int[] a, int start) {
    //Position der kleinsten Karte
    int pos = start;

    // Gehe Array durch und merke die Position der kleinsten Karte ①
    for (int i = start+1; i < a.length; i++) {
        // Wenn eine kleinere Karte gefunden wurde...
        if (a[i] < a[pos]) {

            ... neue Position merken
            pos = i;
        }
    }

    // kleinste Karte mit erster Karte des Reststapels tauschen ② ③
    int temp = a[start]; // erste Karte merken
    a[start] = a[pos];   // kleinste Karte nach vorne bringen
    a[pos] = temp;       // gemerkte Karte in die Mitte des Stapels schreiben

    // Wenn es noch einen Reststapel gibt, soll dieser weitersortiert werden ④
    if (start < a.length) {
         sortiere(a, start+1);
    }
}
```

① Gehe den Stapel durch und merke die Position, an der sich die kleinste Karte befindet.

② Tausche die Position der kleinsten Karte mit der untersten Karte im Stapel.

③ Die kleinste Karte ist jetzt an der richtigen Position.

④ Führe die Schritte nochmal für den Reststapel (ohne die sortierten Karten) aus.