Allgemeines Programmierpraktikum **Sommersemester 2018**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Dominick Leppich Institut für Informatik

Projekt

Abgabe bis 15.07.2018, 23:59 Uhr.

Prüfungen in der Zeit vom 07.08. bis 16.08.2018.

Organisation

1. Wenn Sie an der Prüfung zum Modul *B.Inf.1802: Programmierpraktikum* teilnehmen möchten, müssen Sie sich in **FlexNow** anmelden.

   An- und Abmeldefrist für die Prüfung ist der **15.07.2018**.

2. Bilden Sie Projektgruppen mit vier Teilnehmern, größere Gruppen müssen ausdrücklich genehmigt werden und bekommen zusätzliche Aufgaben.

3. Vereinbaren Sie mit einem Tutor, der Ihr Projekt betreuen soll, einen Termin für ein regelmäßiges Treffen.

4. Bestimmen Sie einen Projektleiter. Zu den Aufgaben des Projektleiters gehört es die Entwicklung des Projekts als Ganzes zu steuern.

   - Werden alle Anforderungen erfüllt?
   - Sind Sprache und Stil der Dokumentation einheitlich?
   - Ist eine überarbeitete Klasse von einem weiteren Projektmitglied kontrolliert worden?
   - ...

5. Geben Sie der Projektgruppe einen aussagefähigen Namen ungleich `FlowerWarsPP`.

6. Der Projektleiter meldet die Projektgruppe an, per E-Mail an `brosenne@informatik.uni-goettingen.de` mit Betreff `FlowerWarsPP`.

   In der E-Mail müssen Projektname, sowie die Namen und Matrikelnummern der Teilnehmer übermittelt werden.

7. Der Projektleiter reserviert einen Prüfungstermin, in Absprache mit den Mitgliedern der Projektgruppe und dem **Tutor**, unter *Terminvergabe→APP-Prüfung* in der Stud.IP-Veranstaltung *Allgemeines Programmierpraktikum* (oder im Profil des Stud.IP-Benutzers *Rechnerübung Informatik*).

   Bei der Reservierung des Prüfungstermins muss der Projektname angegeben werden.

GitLab

1. Für das Projekt wird das GitLab der GWDG `https://gitlab.gwdg.de/` als Versionskontrollsysteme verwendet. Alle Studierenden können den Dienst mit Ihrer E-Mail-Adresse `@stud.uni-goettingen.de` und dem zugehörigen Passwort nutzen.

2. Bestimmen Sie einen aus Ihrer Gruppe zum GitLab-Administrator, der dann für Ihre Gruppe ein neues GitLab-Projekt, unter dem Namen der Projektgruppe, anlegt.

   a) Das Projekt ist **private**, also nur für Mitarbeiter des Projektes zugänglich.

   b) Der GitLab-Administrator lädt alle Gruppenmitglieder ins Projekt ein.

   c) Laden Sie auch Ihren Tutor, Dominick Leppich (`dominick.leppich`) und Henrik Brosenne (`hbrosen`) ins Projekt ein.

Prüfung

Nach der Abgabe wird das Projekt als Ganzes bewertet.

Während der Prüfung stellt jeder Teilnehmer den Teil des Projektes vor, für dessen Implementation er verantwortlich ist. Besonders interessant sind die aufgetretenen Probleme und deren Lösungen.

Neben der korrekten Umsetzung der Projektanforderungen wird gut lesbarer und strukturierter Quellcode erwartet. Es sollten die Grundlagen des objektorientierten Programmentwurfs (z.B. Kapselung, Vererbung, Polymorphismus) berücksichtigt und die Möglichkeiten von Java (z.B. *Java Collections Framework*) ausgenutzt werden.

Jedem Teilnehmer werden Fragen zum Projekt, sowie zu Java, JavaDoc, Git und Ant, im Rahmen von Vorlesung und Übung, gestellt. Daraus ergibt sich für jeden Teilnehmer eine Tendenz (z.B. etwas schwächer als das Projekt insgesamt) und letztlich eine individuelle Note.

FlowerWarsPP

> Die Gebrüder Gartenpfleger Torsten und Torben streiten sich um Tamara und möchten sie durch das Herrichten einer hübschen Gartenlandschaft beeindrucken. Leider müssen sich die beiden die Wiese hierfür teilen. Wer schafft es die meisten Gärten zu bepflanzen und diese zu verzieren indem sie durch Gräben miteinander verbunden werden?

Alle Informationen zu dem Spiel finden Sie auf der GitLab Seite des Projektes.

`https://gitlab.gwdg.de/app/flowerwarspp`

Auf dieser Seite sind folgende Dinge zu finden:

- Spielregeln
- Alle vorgegebenen Klassen des Projektes
- Hilfestellungen zur Implementierung
- Eine Strategie für einen einfachen Computerspieler
- Eine Anleitung zur Nutzung des `ArgumentParser`
- Eine Anleitung zur Nutzung der automatisierten Tests
- Eine Anleitung zur Einbindung des grafischen Board-Viewers

Implementierung

Realisieren Sie **FlowerWarsPP** als Computerspiel in Java.

1. **Allgemein**

   (a) Alle Klassen und Schnittstellen gehören zu einem Package, das mit `flowerwarspp` beginnt.

   (b) Die vorgegebenen Klassen und Schnittstellen des Package `flowerwarspp.preset` dürfen nicht verändert werden.

   Es ist allerdings erlaubt folgende Klassen direkt zu erweitern (nähere Informationen sind dem entsprechenden Abschnitt der Projektbeschreibung zu entnehmen).

   `Viewer` Darf um weitere Funktionen erweitert werden. Vorhandene Funktionen dürfen nicht geändert werden.

   `ArgumentParser` Darf um weitere Parameter ergänzt werden.

   Erweiterung mit Hilfe von Vererbung ist grundsätzlich für alle Klasse, Enumerations und Schnittstellen des Package `flowerwarspp.preset` zulässig.

   Methoden der Spieler-Klassen (siehe *4. Spieler* und *5. Hauptprogramm*) dürfen als Argumente und Rückgabewerte nur Instanzen genau der Klassen aus `flowerwarspp.preset` benutzen. Deshalb sollten Sie, wenn Sie `Move`, `Flower`, `Ditch`, oder `Position` erweitern möchten, nicht Vererbung sondern die Einbettung dieser Klassen in Wrapper- oder Container-Klassen verwenden.

   (c) Kommentieren Sie den Quellcode ausführlich. Verwenden Sie JavaDoc für das *Application Programming Interface (API)* und kommentieren Sie sonst wie üblich.

   (d) Verwenden Sie *Ant* zum automatisierten Übersetzen des Programms und zum Erstellen der Dokumentation.

2. **Spielbrett**

   Erstellen Sie eine Spielbrett-Klasse mit folgenden Merkmalen.

   (a) Implementieren Sie das Interface `flowerwarspp.preset.Board`.

   (b) Ein Spielfeld der Größe n mit $3 \leq n \leq 30$.

   (c) Es muss einen Konstruktor geben, dem nur die Spielfeldgröße als `int` übergeben wird. Das wird benötigt, um das Spielbrett automatisiert testen zu können.

   Hinweis

   Ob Sie alles richtig gemacht haben können Sie leicht testen, indem Sie Ihre Implementation des Spielbretts testen lassen.

   (d) Gültige Spielzüge und Spielzüge die zum Ende des Spiels führen werden erkannt.

   (e) Wenn das Situation auf dem Spielbrett ungültig ist oder das Spiel bereits beendet ist, kann kein weiterer Zug entgegengenommen werden. Wird dennoch versucht einen Zug auszuführen, muss darauf mit einer `IllegalStateException` reagiert werden.

(f) Der erste entgegengenommene Spielzug gehört immer zum roten Spieler.

(g) Ein Spielzug ist ein Objekt der Klasse `flowerwarspp.preset.Move`. Ein Move hat immer eines der nachfolgenden Formate:

- Zwei Referenzen auf Objekte der Klasse `flowerwarspp.preset.Flower`
- Eine Referenz auf ein Objekt der Klasse `flowerwarspp.preset.Ditch`
- Keine Referenzen, dafür aber den Typ `flowerwarspp.preset.MoveType` mit Wert `End`
- Keine Referenzen, dafür aber den Typ `flowerwarspp.preset.MoveType` mit Wert `Surrender`

(h) Die Schnittstelle `flowerwarspp.preset.Viewable` wird implementiert.

3. **Ein- und Ausgabe**

(a) Erstellen Sie eine Klasse, die die Schnittstelle `flowerwarspp.preset.Viewer` implementiert.

Diese Klasse soll es ermöglichen alle für das Anzeigen eines Spielbrett-Objekts nötigen Informationen zu erfragen, ohne Zugriff auf die Attribute des Spielbrett-Objekts zuzulassen.

Die Methode `viewer()` des Spielbretts liefert ein passendes Objekt dieser Klasse. Aus diesem Grund muss die Spielbrettklasse alle notwendige Funktionalität enthalten, um die Funktionen des `flowerwarspp.preset.Viewer` Interfaces umsetzen zu können.

(b) Erstellen Sie eine Text-Eingabe-Klasse, die die Schnittstelle `flowerwarspp.preset.Requestable` implementiert.

Die Methode `request()` fordert einen Zug, in einer Zeile, von der Standardeingabe an und liefert ein dazu passendes `flowerwarspp.preset.Move`-Objekt zurück.

Verwenden Sie die statische Methode `parseMove(String)` der Klasse `Move`, um den von der Standardeingabe eingelesenen String in ein Move-Objekt umzuwandeln.

Die Methode `parseMove` wirft eine `flowerwarspp.preset.MoveFormatException`, falls das Einlesen missglücken sollte. Auf diese Exception muss sinnvoll reagiert werden.

(c) Entwerfen Sie eine Schnittstelle für die Ausgabe des Spielbretts. Verwenden Sie die `flowerwarspp.preset.Viewer` Schnittstelle, um Informationen über das Spielbrett an die anzeigende Klasse weiterzugeben.

Hinweis

Sie können sich hierfür an der Klasse `flowerwarspp.boarddisplay.BoardDisplay` orientieren, die Ihnen auf der GitLab Seite zur Verfügung steht.

(d) Erstellen Sie eine grafische Ein-Ausgabe-Klasse. Diese Klasse implementiert die Schnittstellen `flowerwarspp.preset.Requestable` und die von Ihnen geschriebene Ausgabe-Schnittstelle und benutzt ein Objekt einer Klasse, die die Schnittstelle `flowerwarspp.preset.Viewer` implementiert, für eine einfache grafische Ausgabe.

Sorgen Sie dafür, dass die Darstellung des Spielbretts der Größe des Fensters angepasst ist und beim Verändern der Fenstergröße mitskaliert. Alle Informationen zum Status des Spiels müssen auf der grafischen Ausgabe erkennbar sein (gepflanzte Blumen, gebaute Gräben, Punktestand, Sieger bei Spielende)

Sorgen Sie dafür, dass von der grafischen Eingabe nur gültige Züge zurückgeliefert werden.

Hinweis

Investieren Sie nicht zu viel Zeit in das Design, denn es wird nicht bewertet.

4. **Spieler**

   (a) Alle Spieler implementieren die Schnittstelle `flowerwarspp.preset.Player`. Die Methoden dieser Schnittstelle sind wie folgt zu verstehen.

   `init`

   Initialisiert den Spieler, sodass mit diesem Spieler-Objekt ein neues Spiel mit einem Spielbrett der Größe `size` und der durch den Parameter `color` bestimmten Farbe, begonnen werden kann.
   Die Spielerfarbe ist einer der beiden Werte der Enumeration `flowerwarspp.preset.PlayerColor` und kann die Werte `Red` und `Blue` annehmen.

   `request`

   Fordert vom Spieler einen Zug an.
   Für den Rückgabewert werden nur Objekt von Klassen dem Package `flowerwarspp.preset` verwendet. D.h. es wird ein `Move`-Objekt rückgeliefert, dass selbst nur Referenzen auf Objekte der Klassen `Flower` oder `Ditch` enthält, die ihrerseits nur Referenzen auf Objekte der Klasse `Position` enthalten.
   Hinweis
   Intern kann der Spieler auch mit Objekten von erweiterten `Move`-, `Flower`-, ... Klassen arbeiten.

   `confirm`

   Übergibt dem Spieler im Parameter `status` Informationen über den letzten mit `request` vom Spieler gelieferten Zug.
   Beispiele
   - Gilt `status ==` *eigener Status* und...
     - ...`status == Status.Ok` war der letzte Zug gültig
     - ...`status == Status.RedWin` war der letzte Zug gültig und der rote Spieler hat das Spiel gewonnen
   - Gilt `status !=` *eigener Status* stimmt der Status nicht mit dem spielereigenen Spielbrett überein. Hier muss mit einer entsprechenden Exception reagiert werden!

   `update`

   Liefert dem Spieler im Parameter `opponentMove` den letzten Zug des Gegners und im Parameter `status` Informationen über diesen Zug.
   Hinweis
   Hier gelten die gleichen Beispiele wie auch für `confirm`.

(b) Ein Spieler hat keine Referenz auf das Spielbrett-Objekt des Programmteils, der die Züge anfordert. Trotzdem muss ein Spieler den Spielverlauf dokumentieren, damit er gültige Züge identifizieren kann. Dazu erzeugt jeder Spieler ein eigenes Spielbrett-Objekt und setzt seine und die Züge des Gegenspielers auf diesem Brett.

Daraus können sich Widersprüche zwischen dem Status des eigenen Spielbretts und dem gelieferten Status des Spielbretts des Hauptprogramms ergeben. Das ist ein Fehler auf den mit einer Exception reagiert wird.

(c) Die Methoden der Player-Schnittstelle müssen in der richtigen Reihenfolge aufgerufen werden. Eine Abweichung davon ist ein Fehler auf den mit einer Exception reagiert werden muss.

Ein Spieler wird zu Spielbeginn mit einem Aufruf von `init` initialisiert und durchläuft danach die Methoden `request`, `confirm` und danach `update` bis das Spiel endet. Im Falle eines blauen Spielers beginnt der Spieler mit `update` statt `request`. Der Zeitpunkt des Spielbeginns und eines erneuten Spiels ist für den Spieler nicht ersichtlich, `init` kann zu einem beliebigen Zeitpunkt aufgerufen werden.

(d) Für ein problemloses Netzwerkspiel ist es nötig, dass die Spielerklassen nur `Exception`'s werfen und keine selbst erstellten Klassen, die von dieser erben. An jeder anderen Stelle im Spiel können eigene Exceptions frei erzeugt und geworfen werden.

(e) Erstellen Sie eine Interaktive-Spieler-Klasse, die die Schnittstelle `flowerwarspp.preset.Player` implementiert.

Ein Interaktiver-Spieler benutzt ein Objekt einer Klasse, die das Interface `flowerwarspp.preset.Requestable` implementiert, um Züge vom Benutzer anzufordern.

(f) Erstellen Sie eine Computerspieler-Klasse, die die Spieler-Schnittstelle implementiert und gültige, aber nicht notwendigerweise zielgerichtete, Züge generiert. Dazu wird aus allen aktuell möglichen gültigen Spielzügen zufällig ein Zug ausgewählt.

Hinweis

Java stellt für die Erzeugung von Pseudozufallszahlen die Klasse `java.util.Random` zur Verfügung.

(g) Erstellen Sie einen weiteren Computerspieler, der zielgerichtet, entsprechend der einfachen Strategie, versucht das Spiel zu gewinnen.

Die Strategie ist auf der GitLab Seite beschrieben:
https://gitlab.gwdg.de/app/flowerwarspp/blob/master/specification/simple-strategy.md

(h) Programmieren Sie einen Netzwerkspieler mit dem sie jede Implementation der Schnittstelle `flowerwarspp.preset.Player` einer anderen FlowerWarsPP-Implementation anbieten können.

Falls Sie den Netzwerkspieler im Netzwerk anbieten möchten, läuft die Spiellogik auf einer entfernten FlowerWarsPP-Implementation. Sehen Sie hierfür eine Möglichkeit vor, das Spiel dennoch über die selbst geschriebene Ausgabe-Schnittstelle zu verfolgen.

5. **Hauptprogramm**

   Erstellen Sie eine ausführbare Klasse mit folgender Funktionalität.

   (a) Die Auswahl der roten und blauen Spieler Klassen (Interaktiver Spieler, einer der Computerspieler) und die Größe des Spielbretts soll beim Starten des Programms über die Kommandozeile festgelegt werden können.

   Verwenden Sie zum Einlesen der Kommandozeilenparameter und zum Abfragen der entsprechenden Einstellungen ein Objekt der Klasse `flowerwarspp.preset.ArgumentParser`.

   In der Dokumentation sind nähere Informationen zum Umgang mit dieser Klasse zu finden.

   (b) Ein Spielbrett in Ausgangsposition mit der eingestellten Größe wird initialisiert.

   (c) Zwei Spielerobjekte werden wie eingestellt erzeugt und über Referenzen der `flowerwarspp.preset.Player`-Schnittstelle angesprochen.

   Beide Spieler benutzen dasselbe Objekt einer Klasse, die das `Requestable`-Interface implementiert, um Züge vom Benutzer anzufordern.

   (d) Von den Spieler-Referenzen werden abwechselnd Züge erfragt. Gültige Züge werden bestätigt und dem jeweils anderen Spieler mitgeteilt.

   Den Spielern werden nur Objekt von Klassen und Enumerations aus dem Package `flowerwarspp.preset` übergeben. D.h. z.B. nur `Move`-Objekte, die selbst nur Referenzen auf Objekte der Klassen `Flower` oder `Ditch` enthalten, die ihrerseits nur Referenzen auf Objekte der Klasse `Position` haben.

   (e) Die gültigen Züge werden auf dem Spielbrett ausgeführt.

   (f) Der aktuelle Stand des Spiels (und des Spielbretts) wird über die selbst geschriebene Ausgabe-Schnittstelle ausgegeben.

   (g) Wenn ein Zug zum Spielende führt, macht die Ausgabe eine Meldung darüber.

   (h) Sorgen Sie dafür, dass man das Spiel Computer gegen Computer gut verfolgen kann, verwenden Sie hierfür den Kommandozeilenparameter `-delay`.

   (i) Sehen Sie eine Möglichkeit vor über das Netzwerk zu spielen.

   Ein Netzwerkspiel findet statt, wenn mindestens einer der Spieler den Typ `REMOTE` hat (siehe `flowerwarspp.preset.PlayerType`) oder wenn ein Spieler im Netzwerk angeboten wird.

   Das Hauptspiel behandelt einen Netzwerkspieler über die Schnittstelle `flowerwarspp.preset.Player` wie jeden anderen Spieler auch.

   Sehen Sie im Falle eines `REMOTE`-Spielers eine Möglichkeit vor, diesen zu finden (Name, Host, Port). Dies können Sie zum Beispiel über weitere Kommandozeilenparameter steuern oder interaktiv abfragen.

   Wenn Sie einen Netzwerkspieler anbieten möchten, wählen Sie auch hier eine geeignete Methode den Spielertypen und den Namen einzustellen, unter dem der Spieler an der RMI Registry registriert werden soll.

   Beim Anbieten wird keine Farbe festgelegt, da der Spieler diese Information beim Aufruf von `init` mitgeteilt bekommt.

6. **Optional**

   Bauen Sie das Spiel weiter aus.

   - Laden und Speichern von Spielständen.
   - Implementieren Sie einen Turniermodus, bei dem z.B. zwei Computerspieler mehrere Spiele, mit abwechselnden Farben gegeneinander bestreiten.
   - Erstellen Sie einen weiteren, intelligenteren Computerspieler, z.B. durch die Vorrausberechnung weiterer Züge und/oder einer besser balancierten und/oder erweiterten Bewertung.
   - Erweitern Sie die grafische Ein-Ausgabe-Klasse um mehr Funktionalität (Anzeigen von gültigen Zügen / Anzeige von Feldern, die nicht mehr bepflanzt werden können).
   - ...

Fragen und Antworten

Falls Sie Fragen zu den Spielregeln haben oder Sie vor einer Spielsituation stehen, die mit den Spielregeln nicht eindeutig geregelt sind, nutzen Sie bitte das **Issue-System** von GitLab.

Bei Bedarf werden dann die Regeln um weitere Beispiele erweitert oder ergänzende Informationen zur Verfügung gestellt.

Anforderungen an das fertige Projekt

1. Per E-Mail an `brosenne@informatik.uni-goettingen.de` wird eine Anleitung und ein Archiv (tar, zip, etc.) ausgeliefert.

2. Das Archiv enthält den Quelltext des FlowerWarsPP-Computerspiels, der sich im Rechnerpool des Instituts für Informatik übersetzen und starten lässt.

   Es gibt ein Ant-Buildfile, das eine lauffähige Version des Spiels, gepackt in ein Jar-File, und die vollständige API-Dokumentation erzeugen kann.

3. Die Anleitung beschreibt wie das Archiv zu entpacken ist, der Quelltext übersetzt, die API-Dokumentation erzeugt und das FlowerWarsPP-Computerspiel gestartet wird. Weiterhin wird die Bedienung Ihrer Spielimplementation beschrieben. Geben Sie diese Anleitung im PDF-Format oder als GitLab Flavored Markdown (GFM) ab.

   Hinweis

   Sie müssen in der Anleitung nicht die Spielregeln erneut erklären. Es geht um die Nutzung Ihrer konkreten Implementation!