

Projekt Vorankündigung

Abgabe bis zum **xx.xx.xxxx, xx:xx Uhr**.

Prüfungen in der Zeit vom **xx.xx** bis **xx.xx.xxxx**.

Organisation

1. Wenn Sie an der Prüfung zum Modul *B.Inf.1802: Programmierpraktikum* teilnehmen möchten, müssen Sie sich bis zum **xx.xx.xxxx** in **FlexNow** anmelden.
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 **breaktroughPP**.
6. Melden Sie Ihre Projektgruppe an, unter Angabe des Namens und der Teilnehmer (E-Mail an **brosenne@informatik.uni-goettingen.de**).
7. Wählen Sie für die Prüfung einen Termin im Testzeitraum und lassen Sie sich diesen Termin bestätigen (E-Mail an **brosenne@informatik.uni-goettingen.de**).

Fusionforge

1. Melden Sie sich unter <https://fusionforge.informatik.uni-goettingen.de/> als Benutzer an.
2. Bestimmen Sie einen aus Ihrer Gruppe zum Fusionforge-Administrator, der dann ein neues Fusionforge-Projekt, unter dem Namen der Projektgruppe, für Ihre Gruppe anlegt.

Folgende Einstellungen sind vorzunehmen.

- a) Als Source Code Management (SCM) wählen Sie Subversion (SVN).
- b) Alle Gruppenmitglieder registrieren sich im Projekt (oder werden vom Administrator registriert).
- c) Das SVN-Verzeichnis ist nicht öffentlich lesbar, also nur für Mitarbeiter des Projektes zugänglich.
- d) Registrieren Sie auch Ihren Tutor als Projektmitarbeiter.

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 lesbar und strukturierter Quellcode erwartet. Es sollten die Grundlagen des objektorientierten Programm-entwurfs (z.B. Kapselung, Vererbung, Polymorphismus) berücksichtigt und die Möglich-keiten von Java (z.B. *Java Collections Framework*) ausgenutzt werden.

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

Spiel

Realisieren Sie **TowerWarsPP** als Computerspiel in Java. Kommentieren Sie den Quell-code ausführlich. Verwenden Sie JavaDoc für das *Application Programming Interface (API)* und kommentieren Sie sonst wie üblich. Verwenden Sie Ant zum automatisier-ten Übersetzen des Quelltextes und Erzeugen der Dokumentation.

Die vorgegebenen Quelltexte finden Sie in der Stud.IP-Veranstaltung unter
Dateien→Projekt→towerwarsPP.tgz

Alle vorgegebenen Klassen sind im Anhang zu finden.

1. Alle Klassen und Schnittstellen gehören zu einem Package, das mit **towerwarspp** beginnt.

Die vorgegebenen Klassen und Schnittstellen des Package **towerwarspp.preset** dürfen, mit Ausnahme von Kommentaren, nicht verändert werden.

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

Viewer Darf komplett selbst geschrieben werden. Im Anhang befindet sich ein Viewer, der lediglich als Anregung dienen soll.

ArgumentParser Darf um weitere Parameter ergänzt werden.

2. Erstellen Sie eine Spielbrett-Klasse mit folgenden Merkmalen:

- Ein Spielfeld mit $n \times n$ Feldern wird verwaltet. Dabei gilt $6 \leq n \leq 26$.
- Es wird eine Methode exportiert mit der Spielzüge entgegengenommen werden.
- Gültige Spielzüge und Spielzüge die zum Ende des Spiels führen werden er-kannt.

Ein leerer Zug (**null**) ist gültig und wird als Aufgabe des Spielers gewertet.

- Der erste entgegengenommene Spielzug gehört immer zum roten Spieler.
- Ein Spielzug ist ein Objekt der Klasse `towerwarspp.preset.Move`, das mit Referenzen auf Objekte der Klasse `towerwarspp.preset.Position` arbeitet.
- Es wird eine Methode exportiert, die einen Wert aus der Enumeration `towerwarspp.preset.Status` zurückliefert, über die der Spielstand erfragt werden kann.
- Es wird eine Methode exportiert, die – ausgehend von der aktuellen Spielsituation – alle möglichen gültigen Spielzüge eines Spielers zurückliefert.
- Die Schnittstelle `towerwarspp.preset.Viewable` wird implementiert.

3. Erstellen Sie eine Klasse, die die Schnittstelle `towerwarspp.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.

4. Erstellen Sie eine Klasse mit der der aktuelle Spielstand eines Spielbretts auf die Standardausgabe geschrieben werden kann. Dazu wird ein Objekt einer Klasse, die die Schnittstelle `towerwarspp.preset.Viewer` implementiert, benutzt.

5. Erstellen Sie eine Text-Eingabe-Klasse, die die Schnittstelle `towerwarspp.preset.Requestable` implementiert.

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

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

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

6. Alle Spieler implementieren die folgende Schnittstelle `towerwarspp.preset.Player`.

- Ein Spieler hat keine Referenz auf das Spielbrett-Objekt des Programnteils, 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.

Für ein problemloses Netzwerkspiel ist es nötig, dass alle Exceptions die in innerhalb der Spielerklassen geworfen werden keine Instanzen eigener Exceptionklassen sind. An jeder anderen Stelle im Spiel können eigene Exceptions frei erzeugt und geworfen werden.

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

Die Methoden dieser Schnittstelle sind wie folgt zu verstehen:

request

Fordert vom Spieler einen Zug an.

confirm

Übergibt dem Spieler im Parameter **boardStatus** Informationen über den letzten mit **request** vom Spieler gelieferten Zug.

Beispiele

- Gilt **boardStatus == Status.OK** war der letzte Zug gültig
- Gilt **boardStatus == Status.RED_WIN** war der letzte Zug gültig und der rote Spieler hat das Spiel gewonnen.

update

Liefert dem Spieler im Parameter **opponentMove** den letzten Zug des Gegners und im Parameter **boardStatus** Informationen über diesen Zug.

init

Initialisiert den Spieler, sodass mit diesem Spieler-Objekt ein neues Spiel mit einem Spielbrett der Größe **size × size** und der durch den Parameter **color** bestimmten Farbe, begonnen werden kann.

Die Spielerfarbe ist einer der beiden Werte der Enumeration

towerwarspp.preset.PlayerColor und kann die Werte **RED** und **BLUE** haben.

7. Erstellen Sie eine Interaktive-Spieler-Klasse, die die Schnittstelle **towerwarspp.preset.Player** implementiert.

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

8. Erstellen Sie eine ausführbare Klasse mit folgender Funktionalität.

- 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 **towerwarspp.preset.ArgumentParser**.

Im Anhang findet sich ein kleines kommentiertes Beispiel wie diese Klasse zu verwenden ist. Weitere Parameter dürfen bei Bedarf gerne hinzugefügt werden.

- Ein Spielbrett in Ausgangsposition mit der eingestellten Größe wird initialisiert.

- Zwei Spielerobjekte werden wie eingestellt erzeugt und über Referenzen der Spielerschnittstelle angesprochen.

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

- Von den Spieler-Referenzen werden abwechselnd Züge erfragt. Gültige Züge werden bestätigt und dem jeweils anderen Spieler mitgeteilt.
- Die gültigen Züge werden auf dem Spielbrett ausgeführt.
- Wenn ein Zug zum Spielende führt werden die Spieler darüber informiert und das Programm beendet.

9. 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.

10. Erstellen Sie einen weiteren Computerspieler, der zielgerichtet, entsprechend der unten beschriebenen einfachen Strategie, versucht das Spiel zu gewinnen.
11. Erstellen Sie eine grafische Ein-Ausgabe-Klasse. Diese Klasse implementiert die Schnittstelle `towerwarspp.preset.Requestable` und benutzt ein Objekt einer Klasse, die die Schnittstelle `towerwarspp.preset.Viewer` implementiert, für eine einfache grafische Ausgabe.

Sorgen Sie dafür, dass die Grafik mit unterschiedlichen Spielfeldgrößen sinnvoll skaliert.

Sorgen Sie dafür, dass man das Spiel Computer gegen Computer mit grafischer Ausgabe verfolgen kann, verwenden Sie hierfür den Kommandozeilenparameter `-delay`.

Hinweis

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

12. Programmieren Sie einen Netzwerkspieler mit dem sie jede Implementation der Schnittstelle `towerwarspp.preset.Player` an eine andere TowerWarsPP-Implementation exportieren können.

Im Anhang befinden sich Codebeispiele für den Umgang mit RMI.

Erweitern Sie die ausführbare Klasse um die Möglichkeit Netzwerkspiele zu spielen. Mit den Schaltern `-local` und `-network` soll gesteuert werden ob es sich um ein lokales oder ein Netzwerkspiel handelt.

Sehen Sie im Falle eines Netzwerkspiels geeignete Kommandozeilenparameter vor um zu steuern, ob ein Spieler im Netzwerk angeboten werden oder ein Spieler aus dem Netzwerk gefunden und mit diesem gespielt werden soll.

13. **Optional.** Bauen Sie das Spiel weiter aus.

- Laden und Speichern von Spielständen
- Implementieren Sie einen Turniermodus
- Erstellen Sie einen weiteren, intelligenteren Computerspieler, z.B. durch die Vorrausberechnung weiterer Züge und/oder einer besser balancierten und/oder erweiterten Bewertung.
- ...

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 TowerWarsPP-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 TowerWarsPP-Computerspiel gestartet wird. Weiterhin wird die Bedienung des Spiels beschrieben.

TowerWarsPP

Allgemein

Das Spiel TowerWarsPP ist ein Spiel für zwei Spieler, wobei ein Spieler mit roten und der andere mit blauen Steinen spielt.

TowerWarsPP wird auf einem Spielfeld mit hexagonalen Feldern gespielt, die in einem Parallelogramm mit $n \times n$ Feldern angeordnet sind. Die Zeilen werden durchnummeriert, die Spalten werden mit Buchstaben bezeichnet. Die obere linke Ecke des Spielfeldes ist mit A1 gekennzeichnet.

Startaufstellung

Für n gilt $4 \leq n \leq 26$ und es gibt eine Distanz d mit

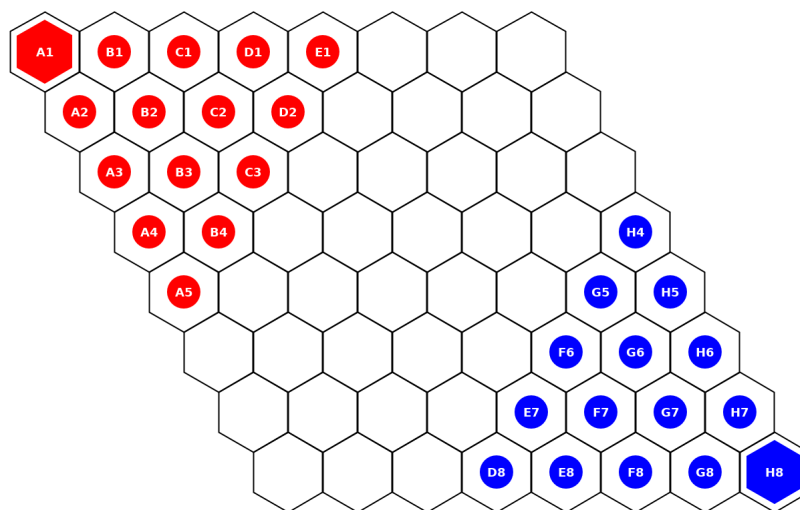
$$d = \left\lfloor \frac{n}{2} \right\rfloor$$

Auf dem Feld A1 steht die rote Basis, auf dem Feld Xn steht die blaue Basis (wobei das X für den n -ten Buchstaben steht).

Alle Felder die einen Abstand $d' \leq d$ zur eigenen Basis haben, werden mit Steinen des jeweiligen Spielers befüllt. Der Abstand ist hierbei so definiert, dass Felder, die sich über eine Seite berühren, den Abstand 1 zueinander haben.

Beispiel

TowerWarsPP-Spielfeld aus 8×8 Feldern mit Startaufstellung.



Ein Nachbar eines Feldes ist ein Feld mit Abstand 1 zu diesem. Jedes Feld in der Mitte hat demnach 6 Nachbarn.

Spielablauf

Der Spieler mit den roten Steinen hat den ersten Zug. Dann wird abwechselnd gezogen, einen Zug auszulassen ist nicht möglich. Ein Feld in TowerWarsPP ist entweder leer oder enthält eines der folgenden Elemente, deren Bedeutung gleich erklärt wird. In jedem Zug muss ein Spieler genau eine Figur von einem auf ein anderes Feld bewegen.

Basis

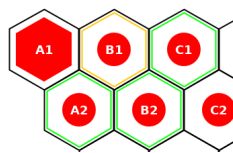
Jeder Spieler hat zu Beginn des Spiels eine Basis, deren Position festgelegt ist. Diese kann nicht bewegt werden. Es ist auch nicht möglich eigene Figuren auf die eigene Basis zu bewegen.

Es ist Ziel des Spiels die gegnerische Basis zu zerstören und folglich auch die eigene zu verteidigen.

Steine

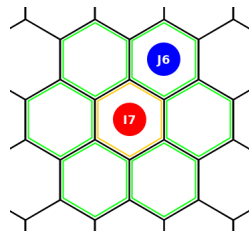
Jeder Spieler hat zu Beginn eine vordefinierte Menge an Steinen.

Ein Stein kann sich von seiner Position aus ein Feld in jede Richtung bewegen (außer auf die eigene Basis natürlich).



B1 kann nicht auf Basis *A1* ziehen

Zieht ein Stein auf ein leeres Feld, ist das ein *Nahzug*. Zieht ein Stein auf ein Feld mit einem gegnerischen Stein, ist dieser geschlagen und der eigene Stein nimmt dessen Platz ein.



I7 kann auf ein benachbartes leeres Feld ziehen oder *J6* schlagen

Ein Stein kann nicht auf die eigene Basis ziehen, sehr wohl jedoch auf die gegnerische Basis (was zum Sieg führt).

Türme

Zu Beginn hat ein Spieler keine Türme. Türme können durch Ziehen eigener Steine auf eigene Steine gebaut und durch Ziehen eigener Steine auf Türme erhöht werden. Es ist durchaus möglich ein Spiel ohne Bauen eines einzigen Turms zu spielen und zu gewinnen.

Türme haben eine Höhe $h \leq h_{\max}$, wobei

$$h_{\max} = \left\lfloor \frac{n}{3} \right\rfloor$$

Beim Erhöhen eines Turms darf die Maximalhöhe nie überschritten werden.

Gegnerische Türme können geschlagen und *blockiert* werden. Außerdem kann die Blockade eigener Türme aufgehoben werden, dazu später mehr.

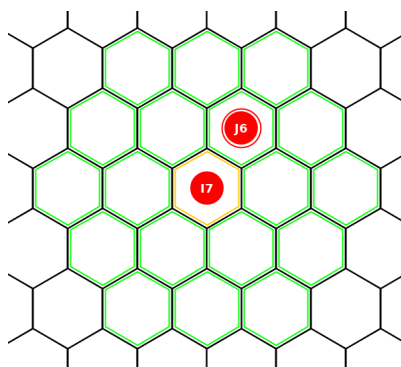
Zieht ein Stein auf ein Feld mit einem eigenen Stein, so wird auf diesem Feld ein Turm der Höhe 1 gebaut.

Zieht ein Stein auf ein Feld mit einem eigenen Turm, so wird dessen Höhe um 1 erhöht (Ausnahme bilden hier blockierte Türme, deren Blockade aufgehoben wird und dessen Höhe unverändert bleibt).

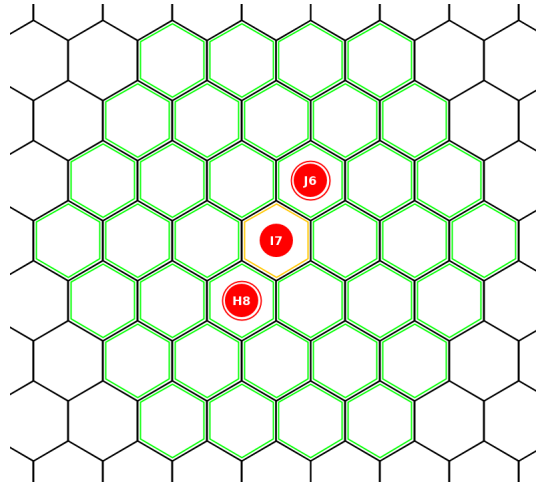
Türme erhöhen die Reichweite angrenzender Steine um die Höhe des Turmes, dabei spielt es keine Rolle ob der Stein in dieser Reichweite Ziehen oder Schlagen will.

Hat ein Stein einen benachbarten Turm, so erhöht sich die Reichweite des Steins um die Höhe h des Turmes. Der Stein kann dann auf alle erlaubten Felder mit Abstand $d \leq 1 + h$ ziehen. Hat ein Stein mehrere angrenzende Türme, so summieren sich deren Höhen bei der Bestimmung der Reichweite auf. Blockierte Türme geben keinen Bonus. Ein Stein mit 6 benachbarten Türmen kann folglich alle Felder mit folgendem Abstand erreichen

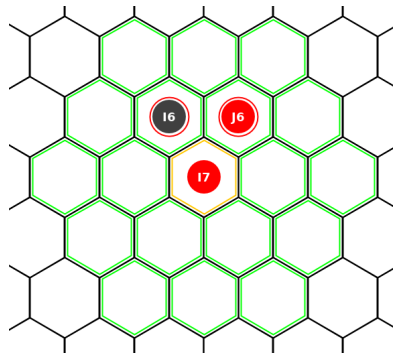
$$d \leq 1 + \sum_{i=1}^6 h_i$$



I7 hat durch den Turm auf *J6* die Reichweite 2

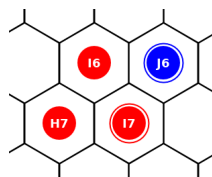


I7 hat durch die Türme auf *J6* und *H8* die Reichweite 3



I7 hat einen Bonus durch Turm *J6* aber nicht durch blockierten Turm *I6*

Zieht ein Stein auf ein benachbartes Feld spricht man von einem *Nahzug*, sonst von einem *Fernzug*.



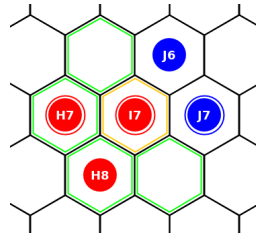
I6 kann gegnerischen Turm *J6* schlagen, *H7* kann den gegnerischen Turm blockieren

Zieht ein Stein auf ein Feld mit einem gegnerischen Turm entscheidet die Art des Zuges was passiert. Handelt es sich um einen *Nahzug*, wird der gesamte gegnerische Turm geschlagen. Im Falle eines *Fernzuges* wird der gegnerische Turm blockiert.

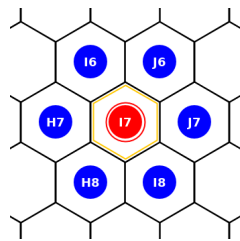
Auch Türme haben Zugmöglichkeiten. Es ist nicht möglich Türme in Ihrer Gesamtheit zu bewegen.

Allerdings kann der oberste Stein vom Turm auf ein benachbartes Feld gezogen werden. Angrenzende Türme verleihen hierfür keinen Bonus auf die Reichweite. Es kann nur auf ein leeres oder eigenes Feld gezogen werden, ohne dabei die Maximalhöhe von Türmen zu überschreiten.

Bei diesem Abbau eines Turmes ist es nicht erlaubt zu schlagen. Daraus ergibt sich direkt, dass Türme nicht die gegnerische Basis schlagen können. Weiter ist es möglich, dass ein Spieler hierdurch keine Handlungsmöglichkeit mehr hat, wenn er nur noch einen Turm besitzt, der komplett von benachbarten gegnerischen Figuren besetzt ist.



- Turm *I7* kann auf ein leeres Feld ziehen
- Turm *I7* kann auf den eigenen Stein *H8* ziehen und damit einen neuen Turm bauen
- Turm *I7* kann auf den eigenen Turm *H7* ziehen und diesen damit erhöhen
- Turm *I7* darf weder nach *J6* noch nach *J7* ziehen, da dort gegnerische Figuren stehen

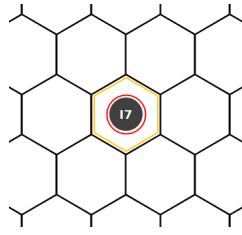


Turm *I7* hat keine Zugmöglichkeiten mehr

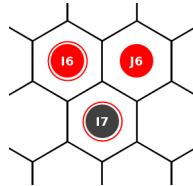
Beim Abbau des Turmes wird dessen Höhe natürlich um 1 reduziert. Wird ein Turm der Höhe 1 abgebaut, so bleibt dort ein normaler Stein zurück.

Ist ein Turm blockiert, kann dieser weder abgebaut werden noch gibt er anderen Steinen einen Bonus. Er steht nur noch blockiert auf dem Feld.

Durch Ziehen eines eigenen Steins auf den Turm kann die Blockade wieder aufgehoben werden, ohne die Höhe des Turms zu verändern. Dies kann durch einen Nah- oder Fernzug oder durch Abbau eines angrenzenden Turms auf diesen Turm erfolgen.



Turm *I7* ist blockiert und kann nicht abgebaut werden



Die Blockade von Turm *I7* kann durch Abbauen des Turms *I6* oder durch den Stein auf *J6* aufgehoben werden

Ende des Spiels

Das Spiel ist gewonnen, wenn eine der folgenden Situationen eintritt:

- Ein Spieler zieht einen Stein auf die gegnerische Basis
- Ein Spieler schlägt alle Steine des Gegnerspieler
- Ein Spieler macht den anderen Spieler handlungsunfähig
- Der Gegenspieler gibt auf

Da in jedem Zug ein Stein bewegt werden muss und im Falle von keiner Zugmöglichkeit der Gegner gewinnt, kann das Spiel nicht unentschieden enden. Es ist jedoch sehr gut möglich, dass ein Spiel endlos andauert.

Spielidee

Die Spielidee für TowerWarsPP wurde dieses Semester von Ole Umlauf und Dominick Leppich entwickelt und im Laufe von Testspielen zu dem ausgebaut, was jetzt in dieser Spielbeschreibung definiert ist.

Einfache Strategie

In Richtung gegnerischer Basis ziehen

Es ist eine gute Idee in Richtung der gegnerischen Basis zu ziehen und weniger gut sich von ihr wegzubewegen. Für jeden Zug sei d_1 der Abstand zur gegnerischen Basis vor dem Zug und d_2 der Abstand nach dem Zug. Daraus berechnet sich ein Basis-Abstandsbonus

$$r_{\text{Basis}} = d_1 - d_2$$

Steine und Türme schlagen

Weiterhin ist es klug gegnerische Einheiten zu schlagen. Berechne hierfür einen Bonus für das Schlagen von gegnerischen Einheiten

$$b_{\text{kick}} = \begin{cases} 0, & \text{wenn keine Einheit geschlagen wurde} \\ 1, & \text{wenn ein Stein geschlagen wurde} \\ 2, & \text{wenn ein Turm geschlagen wurde} \\ 1337, & \text{wenn die gegnerische Basis geschlagen wurde} \end{cases}$$

Gesamtwertung

Berechnen Sie daraus für jeden Zug eine Bewertung

$$\text{score} = r_{\text{Basis}} + b_{\text{kick}}$$

und wählen Sie unter den Zügen mit Maximalbewertung zufällig einen aus.

Zusätzlich gelten die beiden folgenden Regeln.

- a) Gibt es einen Zug mit dem das Spiel gewonnen werden kann, dann wird dieser ausgeführt.
- b) Ein Zug, der zur Folge hätte, dass der Gegner im nächsten Zug gewinnen könnte, wird nur ausgeführt, wenn es keine Alternative gibt.

Quellcode

Preset

Die Klasse Position, die Positionen auf dem Spielbrett beschreibt.

```
1 package towerwarspp.preset;
2
3 import java.io.Serializable;
4
5 public class Position implements Serializable {
6     private int letter, number;
7
8     // -----
9
10    public Position(int letter, int number) {
11        setLetter(letter);
12        setNumber(number);
13    }
14
15    public Position(Position position) {
16        if (position == null)
17            throw new IllegalArgumentException("position_==_null");
18
19        setLetter(position.getLetter());
20        setNumber(position.getNumber());
21    }
22
23    // -----
24
25    public int getLetter() {
26        return letter;
27    }
28
29    private void setLetter(int letter) {
30        if (letter <= 0 || letter > 26)
31            throw new IllegalArgumentException("letter_" + letter + "_out_of_range!");
32
33        this.letter = letter;
34    }
35
36    public int getNumber() {
37        return number;
38    }
39
40    private void setNumber(int number) {
41        if (number <= 0 || number > 26)
42            throw new IllegalArgumentException("number_" + number + "_out_of_range!");
43
44        this.number = number;
45    }
46
47    // -----
48
49    public static Position parsePosition(String str)
50        throws IllegalArgumentException, PositionFormatException {
51
52        if (str == null)
53            throw new IllegalArgumentException("str_==_null");
54
55        try {
56            return new Position(Character.toUpperCase(str.charAt(0)) - 'A' + 1,
57                                Integer.parseInt(str.substring(1)));
58        } catch (IndexOutOfBoundsException | NumberFormatException e) {
59            throw new PositionFormatException("Error_parsing:_" + str + "\", e);
60        }
61    }
62
63    @Override
64    public String toString() {
65        return "" + (char) (getLetter() + 'A' - 1) + getNumber();
```

```

66     }
67
68     @Override
69     public int hashCode() {
70         return Integer.hashCode(getLetter()) ^ Integer.hashCode(getNumber());
71     }
72
73     @Override
74     public boolean equals(Object o) {
75         if (o == null)
76             return false;
77         if (!(o instanceof Position))
78             return false;
79         Position p = (Position) o;
80         return p.getLetter() == getLetter() && p.getNumber() == getNumber();
81     }
82 }

```

Die Klasse Move, die Spielzüge beschreibt.

```

1  package towerwarspp.preset;
2
3  import java.io.Serializable;
4
5  public class Move implements Serializable {
6      private Position start;
7      private Position end;
8
9      // -----
10
11     public Move(Position start, Position end) {
12         setStart(start);
13         setEnd(end);
14     }
15
16     public Move(Move move) {
17         if (move == null)
18             throw new IllegalArgumentException("move_==_null");
19
20         setStart(move.getStart());
21         setEnd(move.getEnd());
22     }
23
24     // -----
25
26     public Position getStart() {
27         return start;
28     }
29
30     private void setStart(Position start) {
31         if (start == null)
32             throw new IllegalArgumentException("start_==_null");
33
34         this.start = start;
35     }
36
37     public Position getEnd() {
38         return end;
39     }
40
41     private void setEnd(Position end) {
42         if (end == null)
43             throw new IllegalArgumentException("end_==_null");
44
45         this.end = end;
46     }
47
48     // -----
49
50     public static Move parseMove(String str) throws IllegalArgumentException,
51                                     MoveFormatException {
52         if (str == null)
53             throw new IllegalArgumentException("str_==_null");

```



```

54
55     // Surrender move
56     if (str.isEmpty())
57         return null;
58
59     try {
60         String[] params = str.split("->");
61         return new Move(Position.parsePosition(params[0]),
62                         Position.parsePosition(params[1]));
63     } catch (IndexOutOfBoundsException | PositionFormatException e) {
64         throw new MoveFormatException("Error parsing: " + str + "\n", e);
65     }
66 }
67
68 @Override
69 public String toString() {
70     return getStart() + "->" + getEnd();
71 }
72
73 @Override
74 public int hashCode() {
75     return getStart().hashCode() ^ getEnd().hashCode();
76 }
77
78 @Override
79 public boolean equals(Object o) {
80     if (o == null)
81         return false;
82     if (!(o instanceof Move))
83         return false;
84     Move m = (Move) o;
85     return m.getStart().equals(getStart()) && m.getEnd().equals(getEnd());
86 }
87 }

```

Exceptions die beim Erzeugen von Positionen und Zügen aus Strings geworfen werden können.

```

1  package towerwarspp.preset;
2
3  public class PositionFormatException extends IllegalArgumentException {
4      public PositionFormatException(String msg) {
5          super(msg);
6      }
7
8      public PositionFormatException(String msg, Throwable cause) {
9          super(msg, cause);
10     }
11 }

```

```

1  package towerwarspp.preset;
2
3  public class MoveFormatException extends IllegalArgumentException {
4      public MoveFormatException(String msg) {
5          super(msg);
6      }
7
8      public MoveFormatException(String msg, Throwable cause) {
9          super(msg, cause);
10     }
11 }

```

Enumeration für Spielerfarbe, Spielertyp und Status.

```
1 package towerwarspp.preset;
2
3 public enum PlayerColor {
4     RED, BLUE
5 }
```

```
1 package towerwarspp.preset;
2
3 public enum PlayerType {
4     HUMAN,
5     RANDOM_AI,
6     SIMPLE_AI,
7     ADVANCED_AI_1,
8     ADVANCED_AI_2,
9     ADVANCED_AI_3,
10    ADVANCED_AI_4,
11    ADVANCED_AI_5
12 }
```

```
1 package towerwarspp.preset;
2
3 import java.io.Serializable;
4
5 public enum Status implements Serializable {
6     OK, RED_WIN, BLUE_WIN, ILLEGAL, UNDEFINED
7 }
```

Die Interfaces zum Kapseln der Ein- und Ausgabe.

```
1 package towerwarspp.preset;
2
3 public interface Requestable {
4     Move deliver() throws Exception;
5 }
```

```
1 package towerwarspp.preset;
2
3 public interface Viewable {
4     Viewer viewer();
5 }
```

```
1 package towerwarspp.preset;
2
3 public interface Viewer {
4     int getSize();
5
6     int getTurn();
7
8     Status getStatus();
9 }
```

Argumentparser

Der ArgumentParser mit seiner Exception.

```
1 package towerwarssp.preset;
2
3 import java.util.HashMap;
4
5 /**
6  * Ein simpler Parser fuer Kommandozeilen Parameter.
7  * <h1>Verwendung</h1>
8  * <p>
9  * Erzeuge innerhalb deiner ausfuehrbaren Klasse eine Instanz dieser Klasse
10 * und uebergib im Konstruktor die Kommandozeilenargumente. Verwende diesen
11 * ArgumentParser um auf Kommandozeilen Parameter zu reagieren.
12 * </p>
13 * <p>
14 * Kommandozeilen Parameter sind entweder Schalter oder Einstellungen.
15 * </p>
16 * <h2>Schalter</h2>
17 * <p>
18 * Ein Schalter ist entweder ein- oder ausgeschaltet. Dementsprechend kann sein
19 * Zustand in einem {@code boolean} abgelegt werden. Schalter sind zu Beginn
20 * ausgeschaltet. Ein Schalter wird ueber den Parameter {@code --SCHALTERNAME}
21 * aktiviert. Ein Schalter kann ueber Kommandozeilen Parameter nicht deaktiviert
22 * werden, da er zu Beginn ohnehin deaktiviert ist.
23 * </p>
24 * <h2>Einstellungen</h2>
25 * <p>
26 * Eine Einstellung hat einen Namen und einen Wert. Ein gutes Beispiel ist hier
27 * die Spielfeldgroesse. Der Name dieser Einstellung ist {@code size}
28 * und der Wert kann eine Zahl zwischen {@code 6} und {@code 26} sein.
29 * Der Typ einer Einstellung richtet sich nach der Einstellung. Die Einstellung
30 * {@code size} zum Beispiel ist ein {@code int}. Einstellungen werden auf der
31 * Kommandozeile mit {@code -NAME WERT} gesetzt.
32 * </p>
33 * <p>
34 * Wird ein Schalter oder eine Einstellung abgefragt die nicht eingelesen wurde,
35 * wird eine {@link ArgumentParserException} geworfen, auf die sinnvoll reagiert
36 * werden muss.
37 * </p>
38 * <p>
39 * Alle Schalter und Einstellungen in dieser Klasse duerfen nicht geaendert
40 * werden. Es ist jedoch erlaubt weitere Schalter oder Einstellungen
41 * hinzuzufuegen,
42 * dies ist im Quellcode kenntlich gemacht.
43 * </p>
44 *
45 * @author Dominick Leppich
46 */
47 public class ArgumentParser {
48     private HashMap<String, Object> params;
49
50     // -----
51
52     /**
53      * Erzeuge einen neuen ArgumentParser aus einem String-Array mit
54      * Parametern. Hier sollte einfach das {@code args}
55      * Argument der {@code main}-Methode weitergereicht werden.
56      *
57      * @param args
58      *         Argumente
59      *
60      * @throws ArgumentParserException
61      *         wenn das Parsen der Argumente fehlschlaegt
62      */
63     public ArgumentParser(String[] args) throws ArgumentParserException {
64         params = new HashMap<>(); parseArgs(args);
65     }
66     // -----
67
68     /**
69      * Parse die Argumente.
```

```

70      *
71      * @param args
72      *      Argumente
73      *
74      * @throws ArgumentParserException
75      *      wenn das Parsen der Argumente fehlschlaegt
76      */
77      private void parseArgs(String[] args) throws ArgumentParserException {
78          // Index to parse
79          int index = 0;
80
81          try {
82              while (index < args.length) {
83                  // Check if argument is a flag or setting
84                  if (args[index].startsWith("--")) {
85                      addFlag(args[index].substring(2)); index += 1;
86                  }
87                  else if (args[index].startsWith("-")) {
88                      addSetting(args[index].substring(1), args[index + 1]);
89                      index += 2;
90                  }
91                  else
92                      throw new ArgumentParserException("Error_parsing:_" +
93                                                          args[index]);
94              }
95              catch (IndexOutOfBoundsException e) {
96                  throw new ArgumentParserException("Missing_parameter");
97              }
98          }
99
100         /**
101          * Fuege einen Schalter hinzu.
102          *
103          * @param flag
104          *      Schalte
105          *
106          * @throws ArgumentParserException
107          *      wenn der Schalter nicht existiert
108          */
109         private void addFlag(String flag) throws ArgumentParserException {
110             // Check if a param with this name already exists
111             if (params.containsKey(flag))
112                 throw new ArgumentParserException("Param_already_exists:_" + flag);
113
114             params.put(flag, new Boolean(true));
115         }
116
117         /**
118          * Fuege eine Einstellung hinzu.
119          *
120          * @param key
121          *      Name
122          * @param value
123          *      Wert
124          *
125          * @throws ArgumentParserException
126          *      wenn die Einstellung nicht existiert oder der Wert ein
127          *      ungueltiges Format hat
128          */
129         private void addSetting(String key, String value) throws
130             ArgumentParserException {
131             // Check if a param with this name already exists
132             if (params.containsKey(key))
133                 throw new ArgumentParserException("Param_already_exists:_" + key);
134
135             if (value.startsWith("-"))
136                 throw new ArgumentParserException("Setting_value_wrong_format:_"
137                                                     + value);
138
139             params.put(key, value);
140         }
141
142         // -----

```

```

143
144 /**
145  * Pruefe ob ein Parameter gesetzt ist.
146  *
147  * @param parameter
148  *      Zu pruefender Parameter
149  *
150  * @return wahr, wenn der Parameter gesetzt wurde
151  */
152 public boolean isSet(String parameter) {
153     return params.containsKey(parameter);
154 }
155
156 /**
157  * Gib den Wert eines Schalters zurueck.
158  *
159  * @param flag
160  *      Name des Schalters
161  *
162  * @return Wert
163  *
164  * @throws ArgumentParserException
165  *      wenn der Schalter den falschen Typ hat (falls eine Einstellung
166  *      versucht wird als Schalter auszulesen)
167  */
168 private boolean getFlag(String flag) throws ArgumentParserException {
169     if (!params.containsKey(flag))
170         return false;
171
172     Object o = params.get(flag); if (!(o instanceof Boolean))
173         throw new ArgumentParserException("This is not a flag");
174
175     return (Boolean) params.get(flag);
176 }
177
178 /**
179  * Gib den Wert einer Einstellung als {@link Object} zurueck.
180  *
181  * @param key
182  *      Name der Einstellung
183  *
184  * @return Wert als {@link Object}.
185  *
186  * @throws ArgumentParserException
187  *      wenn die Einstellung nicht existiert
188  */
189 private Object getSetting(String key) throws ArgumentParserException {
190     if (!params.containsKey(key))
191         throw new ArgumentParserException("Setting " + key + " not " +
192             "defined");
193
194     return params.get(key);
195 }
196
197 // -----
198
199 /**
200  * Interpretiere einen Spielertypen
201  *
202  * @param type
203  *      Eingelesener Typ
204  *
205  * @return Spielertyp als {@link PlayerType}
206  *
207  * @throws ArgumentParserException
208  *      wenn der eingelese Typ nicht passt
209  */
210 private PlayerType parsePlayerType(String type) throws
211     ArgumentParserException {
212     switch (type) {
213         case "human": return PlayerType.HUMAN;
214         case "random": return PlayerType.RANDOM_AI;
215         case "simple": return PlayerType.SIMPLE_AI;

```

```

216         case "adv1": return PlayerType.ADVANCED_AI_1;
217         case "adv2": return PlayerType.ADVANCED_AI_2;
218         case "adv3": return PlayerType.ADVANCED_AI_3;
219         case "adv4": return PlayerType.ADVANCED_AI_4;
220         case "adv5": return PlayerType.ADVANCED_AI_5;
221
222         default: throw new ArgumentParserException("Unknown player type: " +
223                                                     "" + type);
224     }
225 }
226
227 // -----
228
229 public boolean isLocal() throws ArgumentParserException {
230     return getFlag("local");
231 }
232
233 public boolean isNetwork() throws ArgumentParserException {
234     return getFlag("network");
235 }
236
237 public boolean isGraphic() throws ArgumentParserException {
238     return getFlag("graphic");
239 }
240
241 public int getSize() throws ArgumentParserException {
242     return Integer.parseInt((String) getSetting("size"));
243 }
244
245 public int getDelay() throws ArgumentParserException {
246     return Integer.parseInt((String) getSetting("delay"));
247 }
248
249 public PlayerType getRed() throws ArgumentParserException {
250     return parsePlayerType((String) getSetting("red"));
251 }
252
253 public PlayerType getBlue() throws ArgumentParserException {
254     return parsePlayerType((String) getSetting("blue"));
255 }
256
257 // *****
258 // Hier koennen weitere Schalter und Einstellungen ergaenzt werden...
259 // *****
260
261 }

```

```

1 package towerwarssp.preset;
2
3 public class ArgumentParserException extends Exception {
4     public ArgumentParserException() {
5
6     }
7
8     public ArgumentParserException(String msg) {
9         super(msg);
10    }
11
12    public ArgumentParserException(String msg, Throwable cause) {
13        super(msg, cause);
14    }
15 }

```

Ein Beispiel zur Verwendung des Parsers.

```
1 package towerwarspp.demo;
2
3 import towerwarspp.preset.ArgumentParser;
4 import towerwarspp.preset.ArgumentParserException;
5
6 public class ArgumentParserTest {
7     public static void main(String[] args) {
8         try {
9             ArgumentParser ap = new ArgumentParser(args);
10
11             System.out.println("local:␣" + ap.isLocal());
12             System.out.println("network:␣" + ap.isNetwork());
13             System.out.println("size:␣" + ap.getSize());
14
15             switch (ap.getRed()) {
16                 case HUMAN:
17                     System.out.println("Red␣chose␣human␣player");
18                     break;
19                 case RANDOM_AI:
20                     System.out.println("Red␣chose␣random␣player");
21                     break;
22
23                 // and so on..
24             }
25         } catch (ArgumentParserException e) {
26             // Something went wrong...
27             e.printStackTrace();
28         }
29     }
30 }
```

Netzwerk (RMI)

Ein Gerüst für einen Netzwerkspieler finden Sie nachfolgend.

```
1 package towerwarspp.demo;
2
3 import java.rmi.*;
4 import java.rmi.server.UnicastRemoteObject;
5 import java.net.*;
6
7 import towerwarspp.preset.*;
8
9 public class NetPlayer extends UnicastRemoteObject implements Player {
10     private Player player;
11
12     // -----
13
14     public NetPlayer(Player player) throws RemoteException {
15         this.player = player;
16     }
17
18     public Move request() throws Exception, RemoteException {
19         return player.request();
20     }
21
22     public void update(Move opponentMove, Status boardStatus)
23         throws Exception, RemoteException {
24         player.update(opponentMove, boardStatus);
25     }
26
27     public void confirm(Status boardStatus)
28         throws Exception, RemoteException {
29         player.confirm(boardStatus);
30     }
31
32     public void init(int size, PlayerColor color)
33         throws Exception, RemoteException {
34         player.init(size, color);
35     }
36 }
```

Die folgende Methode `offer` ist ein Beispiel wie ein Netzwerkspieler an einen Rechner `host`, auf dem eine RMI-Registry läuft, unter einem beliebigen Namen `name` angeboten werden kann.

```
1     public void offer(Player p, String host, String name) {
2         try {
3             Naming.rebind("rmi://" + host + "/" + name, p);
4             System.out.println("Player_" + name + ")_ready");
5         } catch (MalformedURLException ex) {
6             ex.printStackTrace();
7         } catch (RemoteException ex) {
8             ex.printStackTrace();
9         }
10    }
```

Die folgende Methode `find` ist ein Beispiel wie ein Spieler auf einen Rechner `host`, auf dem eine RMI-Registry läuft, unter einem bekannten Namen `name` gefunden werden kann.

```
1     public Player find(String host, String name) {
2         Player p = null;
3         try {
4             p = (Player) Naming.lookup("rmi://" + host + "/" + name);
5             System.out.println("Player_" + name + ")_found");
6         } catch (Exception ex) {
7             ex.printStackTrace();
8         }
9         return p;
10    }
```
