Allgemeines Programmierpraktikum **Sommersemester 2019**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Institut für Informatik

Projekt

Abgabe bis 21.07.2019, 23:55 Uhr.

Prüfungen in der Zeit vom 19.08. bis 29.08.2019.

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 **15.07.2019**.

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 `Nowhere2GoPP`.

6. Der Projektleiter meldet die Projektgruppe an, per E-Mail mit Betreff `Nowhere2GoPP` an `brosenne@informatik.uni-goettingen.de`.

   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 und Henrik Brosenne (`hbrosen`) ins Projekt ein.

3. Auf der GitLab Seite des Projektes `https://gitlab.gwdg.de/app/nowhere2gopp` finden Sie die Klassen, Enumerations und Schnittstellen aus dem Abschnitt Implementierung

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.

Implementierung

Realisieren Sie **Nowhere2GoPP** (siehe Spielregeln) als Computerspiel in Java.

1. **Allgemein**

   a) Alle Klassen und Schnittstellen gehören zu einem Package, das mit `nowhere2gopp` beginnt.

   b) Die vorgegebenen Klassen und Schnittstellen des Package `nowhere2gopp.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 `nowhere2gopp.preset` zulässig.

   Methoden der Spieler-Klassen (siehe Spieler und Hauptprogramm) dürfen als Argumente und Rückgabewerte nur Instanzen genau der Klassen aus `nowhere2gopp.preset` benutzen. Deshalb sollten Sie, wenn Sie `Move`, `Site`, etc. 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) Die Schnittstelle `nowhere2gopp.preset.Playable` wird implementiert.

   b) Ein Spielfeld hat die Größe $n = 2k + 1$ für $1 \leq k \leq 5$.

   c) Es gibt einen Konstruktor, dem nur die Spielfeldgröße als `int` übergeben wird.

   Hinweis. Wird benötigt, um das Spielbrett automatisiert testen zu können.

   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 `nowhere2gopp.preset.Move`. Ein Move hat immer eines der nachfolgenden Formate.

- Den Typ `nowhere2gopp.preset.MoveType.End` und keine Referenzen.
- Den Typ `nowhere2gopp.preset.MoveType.Surrender` und keine Referenzen.
- Den Typ `nowhere2gopp.preset.MoveType.LinkLink` und zwei Referenzen auf Objekte der Klasse `nowhere2gopp.preset.SiteSet`. Für Züge der erste Spielphase.
- Den Typ `nowhere2gopp.preset.MoveType.AgentLink` und jeweils eine Referenz auf ein Objekt der Klasse `nowhere2gopp.preset.SiteTuple` und `nowhere2gopp.preset.SiteSet`. Für Züge der zweiten und dritten Spielphase.

h) Die Schnittstelle `nowhere2gopp.preset.Viewable` wird implementiert.

3. **Ein- und Ausgabe**

a) Erstellen Sie eine Klasse, die die Schnittstelle `nowhere2gopp.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 `nowhere2gopp.preset.Viewer` Interfaces umsetzen zu können.

b) Erstellen Sie eine Text-Eingabe-Klasse, die die Schnittstelle `nowhere2gopp.preset.Requestable` implementiert.

Die Methode `request()` fordert einen Zug, in einer Zeile, von der Standardeingabe an und liefert ein dazu passendes `nowhere2gopp.preset.Move`-Objekt zurück.

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

Die Methode `parse` wirft eine `nowhere2gopp.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 `nowhere2gopp.preset.Viewer` Schnittstelle, um Informationen über das Spielbrett an die anzeigende Klasse weiterzugeben.

d) Erstellen Sie eine grafische Ein-Ausgabe-Klasse. Diese Klasse implementiert die Schnittstellen `nowhere2gopp.preset.Requestable` und die von Ihnen geschriebene Ausgabe-Schnittstelle und benutzt ein Objekt einer Klasse, die die Schnittstelle `nowhere2gopp.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 (z.B. Sieger bei Spielende)

Hinweis

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

4. **Spieler**

a) Alle Spieler implementieren die Schnittstelle `nowhere2gopp.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 `nowhere2gopp.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 aus dem Package `nowhere2gopp.preset` verwendet. D.h. es wird ein `Move`-Objekt rückgeliefert.

Hinweis

Intern kann der Spieler bei Bedarf auch mit Objekten von Erweiterungen der Klassen `Move`, `Site`, etc. 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) Erstellen Sie eine Interaktive-Spieler-Klasse, die die Schnittstelle `nowhere2gopp.preset.Player` implementiert.

   Ein Interaktiver-Spieler benutzt ein Objekt einer Klasse, die das Interface `nowhere2gopp.preset.Requestable` implementiert, um Züge vom Benutzer anzufordern.

e) 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.

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

   Die einfache Strategie wird in einem separaten Dokument nachgereicht.

g) Programmieren Sie einen Netzwerkspieler mit dem sie jede Implementation der Schnittstelle `nowhere2gopp.preset.Player` einer anderen Nowhere2GoPP-Implementation anbieten können.

   Falls Sie den Netzwerkspieler im Netzwerk anbieten möchten, l äuft die Spiellogik auf einer entfernten Nowhere2GoPP-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 `nowhere2gopp.preset.ArgumentParser`.

      Im Quellcode der Klasse sind nähere Informationen zum Umgang mit dieser Klasse zu finden.

      Danke an Dominick Leppich, den Autor der Klasse.

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

   c) Zwei Spielerobjekte werden wie eingestellt erzeugt und über Referenzen der `nowhere2gopp.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 `nowhere2gopp.preset` übergeben.

   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 `nowhere2gopp.preset.PlayerType`) oder wenn ein Spieler im Netzwerk angeboten wird.

      Das Hauptspiel behandelt einen Netzwerkspieler über die Schnittstelle `nowhere2gopp.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 beimc Aufruf von `init` mitgeteilt bekommt.

6. **Optional**

   Bauen Sie das Spiel weiter aus.

   - Laden und Speichern von Spielständen.
   - Implementieren Sie einen Tournament-Modus, bei dem zwei Computerspieler mehrere (viele) Spiele, in unterschiedlichen Konstellationen (z.B. mit wechselnden Farben) gegeneinander bestreiten. Erstellen Sie eine Statistik der Spielverläufe und -ergebnisse.
   - Entwickeln 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 (z.B. Anzeigen von gültigen Zügen).
   - ...

Spielregeln

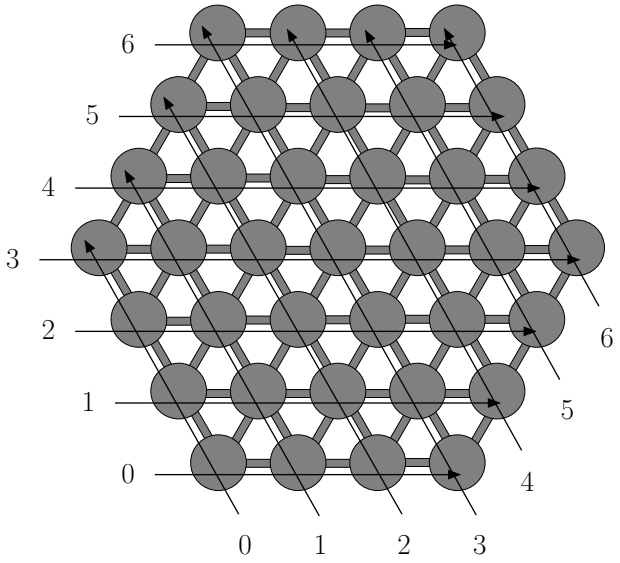
Allgemein

Die „Nowhere to Go“-Variante des Allgemeinen Programmierpraktikums (Nowhere2GoPP) ist ein Spiel für zwei Spieler, Rot und Blau.

Nowhere2GoPP wird auf einem sechseckigen Spielfeld der Größe $n = 2k + 1$ für $1 \leq k \leq 5$ gespielt. Ein Spielfeld besteht aus n Zeilen und n Spalten (oder Diagonalen) von Plattformen, die jeweils von 0 bis $n - 1$ durchnummeriert sind. Zeile und Spalten mit derselben Nummer bestehen aus derselben Anzahl von *sites*. Die Zeile/Spalte k besteht aus n *sites*, die Zeile/Spalte $i - 1$ und $n - i$ mit $1 \leq i \leq k$ jeweils aus $k + i$ *sites*. In der Ausgangsposition sind alle benachbarte *sites* paarweise über *links* verbunden.

Beispiel

Nowhere2GoPP-Spielfeld der Größe 7 in der Ausgangsposition mit Nummerierung der Zeilen und Spalten.



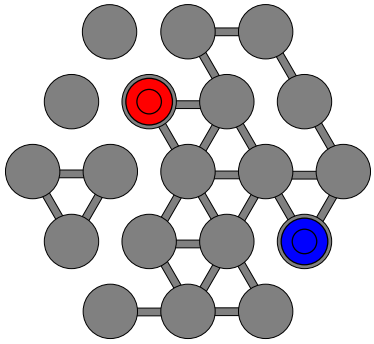
Jeder Spieler hat eine Spielfigur (*agent*) in seiner Farbe. Im Folgenden wird der *agent* der Spielers Rot mit *red* und der des Spielers Blau mit *blue* bezeichnet. Nachdem die *agents* ins Spiel gekommen sind, besetzt jeder *agent* genau eine *site* des Spielfelds. Weiterhin haben die Spieler die Möglichkeit *links* zu entfernen.
Die Position einer *site* ist über das Tuple $(\text{Spalte}, \text{Zeile})$ eindeutig bestimmt. Ein *link* ist durch die beiden *sites*, die er verbindet, eindeutig bestimmt, deshalb wird ein *link* über eine Menge von *site*-Positionen $\{(\text{Spalte}_s, \text{Zeile}_s), (\text{Spalte}_t, \text{Zeile}_t)\}$ beschrieben.

Hinweis

Diese Darstellung erlaubt jeweils zwei Möglichkeiten einen *link* zu beschreiben, $\{(x_s, y_s), (x_t, y_t)\} = \{(x_t, y_t), (x_s, y_s)\}$.

Beispiel

Spielfeld der Größe 5 mit *red* auf *site* $(2, 3)$ und *blue* auf $(3, 1)$, sowie einigen entfernten *links*, z.B. $\{(0, 0), (0, 1)\}$ und $\{(2, 4), (1, 3)\}$.



Spielablauf

Das Spiel startet mit einem leeren Spielfeld, auf dem alle *sites* und *links* vorhanden sind.

Rot hat den ersten Zug. Dann wird abwechselnd gezogen, einen Zug auszulassen ist nicht möglich. Ein Zug umfassend, abhängig von der aktuellen Spielphase, die Aktionen Bewegen und Entfernen.

Bewegen

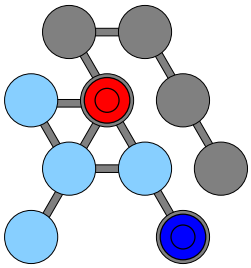
Ein *agent* kann sich von der *site* X, auf der er sich befindet, auf einen beliebige nicht blockierte *site* Y bewegen, wenn es einen Weg zwischen X und Y gibt, der nur über (vorhandene) *links* und nicht blockierte *sites* führt. Wobei eine *site* blockiert ist, wenn sich ein *agent* auf ihr befindet.

Entfernen

Grundsätzlich kann jeder (vorhandene) Link entfernt werden.

Beispiel

Der *agent blue* kann sich nur auf die hellblau markierten Felder bewegen, wohingegen sich *red* auf jede, bis auf die von *blue* blockierte, *site* bewegen kann.



Es gibt drei **Spielphasen**.

1. In 2^{k-1} Runden entfernt jeder Spieler pro Zug zwei *links*.
2. Jeder Spieler hat einen Zug, in dem er seinen *agent* auf dem Spielfeld platziert, den *agent* bewegt und anschließend einen *link* entfernt.
3. Pro Zug bewegt ein Spieler seinen *agent* und entfernt anschließend einen *link*.

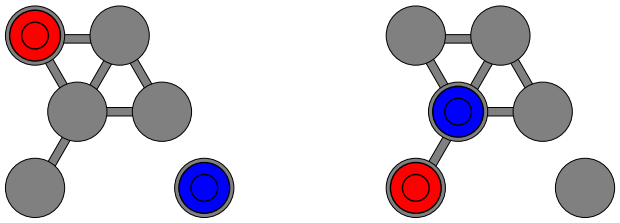
Hinweise

- Bei einer Bewegung muss der *agent* die *site*, auf der er sich befindet, verlassen.
- Es ist nicht möglich eine der Aktionen auszulassen oder nur teilweise (z.B. nur Bewegen) auszuführen.

Ende des Spiels

Das Spiel ist entschieden, wenn der Spieler, der am Zug ist, keinen vollständigen Zug ausführen kann, d.h. in der Regel, dass der *agent* keinen Möglichkeit hat sich zu bewegen. In diesem Fall hat der Spieler, der am Zug ist, das Spiel verloren.

Beispiele



a) b)

a) Spieler Blau ist am Zug, *blue* hat keine Möglichkeit, die *site*, auf der er sich befindet, zu verlassen, weil es keine *links* mehr gibt, die diese *site* mit einer anderen verbinden. Deshalb hat Blau das Spiel verloren.

b) Spieler Rot ist am Zug, *red* hat keine Möglichkeit, die *site*, auf der er sich befindet, zu verlassen, weil alle Wege zu anderen *sites* über die von *blue* blockierte *site* führen. Deshalb hat Rot das Spiel verloren.