Praktisches Projekt: The King of the Hill

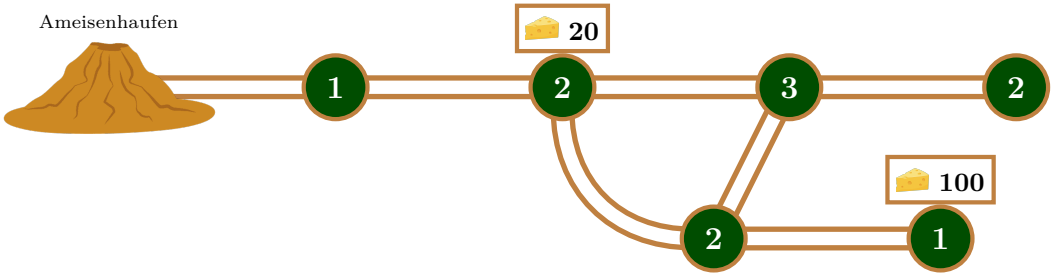
Hinweis: Falls Sie Meilenstein A nutzen möchten, müssen Sie diesen bis 26.07.2021 23:59 abgeben. Die Abgabefrist für Meilenstein Ω ist 20.08.2021 23:59.

1 Projektbeschreibung

Neueste Forschungsergebnisse des Zentrums für Bioinformatik Saarbrücken haben ergeben, dass das Konzept der Nebenläufigkeit nicht menschengemacht ist, sondern bereits vielfältig in der Natur verbreitet ist. So konnte bei einer seltenen Ameisenspezies, den Concurants, nicht nur eindeutig nebenläufiges, sondern auch noch korrektes nebenläufiges Verhalten bei der Futtersuche beobachtet werden.

Da die Spezies der Concurants vom Aussterben bedroht ist, sind weitere Experimente mit den Tieren selbstverständlich verboten. Glücklicherweise findet sich in der Vorlesung *Nebenläufige Programmierung* eine Gruppe Studierender, die quasi dazu prädestiniert ist das Verhalten der Ameisen zu simulieren, damit weitere Untersuchungen gemacht werden können ohne den Bestand dieser seltenen Spezies zu gefährden.

Der Leiter des Zentrum für Bioinformatik, Reiner-Theo Retiker, erstattet Ihnen von seinen bisherigen Forschungsergebnissen enthusiastisch Bericht:

> Die Concurants begreifen ihre Umgebung als Graph.
>
> Die Wege bilden die Kanten des Graphen. Auf jedem Weg darf sich jeweils nur eine Ameise befinden. Allerdings sind die Concurants wahre Verbiegungskünstler und können sich auch bei jedem noch so kleinen Weg aneinander vorbeiquetschen, wenn sie sich in die entgegengesetzte Richtung bewegen. (D.h. konkret, dass auf jedem Weg nur eine Ameise in jede Richtung vorhanden sein kann.)
>
> Bei den *Knoten* des Graphen kann es sich um drei verschiedene Elemente handeln:
>
> (a) *Lichtungen* verknüpfen mindestens einen, meist aber mehrere Wege. Außerdem haben Lichtungen durch ihre geografische Gegebenheit möglicherweise nur Platz für eine begrenzte Anzahl an Ameisen. Dieser Platz ist für jede Lichtung unterschiedlich.
>
> (b) *Futterquellen* sind Lichtungen auf denen sich *Futter* befindet. Futterquellen sind nicht unendlich, sondern haben eine Kapazität. Ist das Futter an einer Futterquelle komplett aufgebraucht, verhält sich diese nur noch wie eine gewöhnliche Lichtung.
>
> (c) *Ameisenhaufen* bieten so viel Platz, dass *quasi* unendlich viele Concurants darin Platz finden. Wegen der Seltenheit dieser Spezies können wir davon ausgehen, dass nur ein Ameisenhaufen im Graph zu finden ist. Der Ameisenhaufen ist durch mindestens einen Weg mit dem Graphen der Lichtungen verbunden.
>
> 
>
>
> Die Skizze zeigt einen kleinen Ausschnitt einer Concurants Umgebung. In dieser befindet sich ein Ameisenhaufen, vier Lichtungen mit unterschiedlich viel Platz (1, 3, 2 und 2) für Concurants, sowie zwei Futterquellen. Die erste Futterquelle bietet Platz für 2 Concurants und besitzt 20 Einheiten Futter, während die zweite Futterquelle nur Platz für eine Concurant hat, aber dafür 100 Einheiten Futter beinhaltet.

Damit die Forschung an den Concurants stattfinden kann, müssen Sie auch das Verhalten der Concurants simulieren. Dazu berichtet Reiner-Theo von seinen **empirischen Beobachtungen**:

> Concurants sammeln das gesamte Futter in der Umgebung und bringen es zu ihrem Ameisenhaufen.
>
> Jede Concurant kann maximal eine Einheit Futter tragen.
>
> Concurants können nicht direkt untereinander kommunizieren. Sie kommunizieren lediglich durch den Einsatz von *Pheromonen*. Diese Pheromone können sehr spezifisch platziert werden. So kann eine Ameise in einem Knoten jeden ausgehenden Weg unterschiedlich markieren (konkret markiert eine Ameise also in einem Knoten k einen Weg p; damit kann der Weg von beiden Enden aus markiert werden). Eine Ameise kann Pheromone natürlich nur in dem Knoten wahrnehmen, in dem sie sich aktuell befindet.
>
> Dabei ist die Besonderheit der Concurants, dass sie sowohl während des Traversieren des Umgebungsgraphen, als auch bei der Kommunikation mittels Pheromonen korrekt nebenläufig arbeiten.
>
> Concurants können gefressen werden. Jede Concurant hat eine unterschiedlich gute Tarnfähigkeit t. Je länger eine Concurant in einem Weg verweilt, desto mehr macht sie auf sich aufmerksam. Sobald sie länger verweilt als ihre Tarnfähigkeit das zulässt (also Verweilzeit $> t$)[1], wird sie entdeckt und gefressen. Eventuell gehaltenes Futter wird mitgefressen, geht also verloren. In einer Lichtung hingegen sind die Concurants vor ihren natürlichen Fressfeinden geschützt[2].

Das Verhalten der Concurants ist noch nicht so weitgehend erforscht, wie Reiner-Theo das gerne hätte. Er macht folgende **Annahmen**, auf deren Basis Sie die Futtersuche simulieren sollen.

> **Das klassische Ameisenverhalten.** Beim klassischen Ameisenverhalten werden 2 Pheromone genutzt: das *Futter-Pheromon* P_F, sowie das *Ameisenhaufen-Pheromon* P_A. Beide Pheromone werden fast ausschließlich als natürliche Zahlen simuliert. Dabei nehmen wir folgenden Wertebereich an:
>
> $P_F \in \{NaP \text{ (not a pheromone)}, MaP \text{ (maximal pheromone)}\} \cup \mathbb{N}$, $P_A \in \{NaP \text{ (not a pheromone)}\} \cup \mathbb{N}$.
>
> Es gilt $\forall n \in \mathbb{N} : n < NaP$, $n < MaP$.
>
> Wir nehmen für die **klassische Futtersuche** der Concurants, inspiriert von gewöhnlichen Ameisen, folgendes Verhalten an:
>
> (a) Bei der Suche nach Futter folgen Ameisen immer dem *geringsten* Futter-Pheromon P_F. Sollten mehrere Futter-Pheromone den gleichen Wert haben, so entscheiden sich Concurants immer *zufällig*. Dies gilt auch für den Fall, dass alle Futter-Pheromone den Wert NaP haben.
>
> Jede der Concurants hat einen unterschiedlichen Grad der *Ungeduld* u. Wenn das geringste Futter-Pheromon P_F einen größeren Wert hat als dieser Grad der Ungeduld (also $u < P_F$), dann entscheidet sich die Ameise stattdessen für einen *zufälligen* Weg mit dem Wert NaP. Dies geht natürlich nur, wenn es mindestens einen Weg mit dem Wert NaP gibt.
>
> In beiden Fällen allerdings nehmen Concurants **niemals** denjenigen Weg, von dem aus der Knoten betreten wurde oder einen Weg, der entweder während der aktuellen Suche schon einmal ausprobiert worden ist oder mit MaP gekennzeichnet wurde.
>
> (b) Wann immer eine Concurant einen mit NaP beschrifteten Weg nimmt, betrachtet sie sich selbst als *Abenteurer*. Sie tut das solange, bis sie via klassischem Rückweg den Ameisenhaufen wieder erreicht hat.
>
> (c) Concurants haben ein sehr gutes Gedächtnis und können sich folgendes merken:
>
> (i) die *Sequenz*, also den gesamten Weg vom Ameisenhaufen bis zum aktuellen Knoten, sowie
>
> (ii) für jeden Knoten einzeln die Information, entlang welcher anliegender Wege sie sich in der aktuellen Suche bereits bewegt haben.
>
> Um das Gedächtnis der Concurants nicht zu überfordern, wird diese gesamte Information vergessen, sobald der Ameisenhaufen via des klassischen Rückweges wieder erreicht wurde.
>
> (d) Wenn eine Concurant einen Knoten betritt, den sie in der direkt zurückliegenden Sequenz schon einmal betreten hat, dann kehrt sie sofort um und geht zum letzten Knoten zurück. Der Knoten wird dann auch sofort wieder aus der Sequenz entfernt.

[1] In der Praxis realisieren wir das durch die Verwendung von (a)wait mit timeouts. Dadurch entstehende Ungenauigkeiten werden ignoriert.

[2] Der einzige natürliche Fressfeind der Concurants, das Sequensquirrel, leidet unter Agoraphobie und meidet deshalb Lichtungen.

(e) Wenn es keinen Weg mehr gibt, den die Concurant nehmen könnte (weil es nur den Eingangsweg gibt, oder alle anderen Wege entweder bereits in der aktuellen Suche von der Concurant ausprobiert worden sind oder mit *MaP* gekennzeichnet sind), dann kehrt die Concurant zu dem vorherigen Knoten der Sequenz zurück. Sie tut das, indem sie den zuvor genommenen Weg rückwärts abläuft. Der dann wieder betretene Knoten gilt jetzt wieder als letzter Knoten der Sequenz und der zuletzt erfolglos besuchte Knoten wird aus der Sequenz gelöscht. Dabei markiert die Concurant diese erfolglos genommene Kante mit $P_F = MaP$; immerhin ist dort kein Futter mehr zu erwarten.

(f) Wenn eine Concurant im eigenen Ameisenhaufen keine Optionen mehr hat (also alle Wege wurden in der aktuellen Suche bereits ausprobiert oder sind mit *MaP* gekennzeichnet), dann nimmt die Concurant an, dass es nichts mehr zu tun gibt und bleibt im Ameisenhaufen (in der Simulation terminiert sie).

(g) Wann immer eine Concurant aus dem Knoten k_1 einen Weg p zum Knoten k_2 genommen hat, aktualisiert diese danach das Ameisenhaufen-Pheromon nach folgender Regelung:

$$P_A(k_2, p^{-1}) = \min\left\{P_A(k2, p^{-1}), w\right\},$$

wobei w die Wegeslänge vom Ameisenhaufen bis zum aktuellen Knoten k_2, berechnet anhand der aktuellen Sequenz, und p^{-1} der umgekehrte Weg (also von k_2 nach k_1), ist.

(h) Sobald eine Concurant eine Futterquelle gefunden hat, nimmt sie eine Einheit des Futters auf und macht sich auf den Rückweg zum Ameisenhügel.

Für den **klassischen Rückweg** nehmen wir folgendes Verhalten an:

(a) Eine Concurant, die sich als *Abenteurer* versteht, nimmt immer exakt den umgekehrten Weg (also die Sequenz), den sie gekommen ist. Ansonsten folgt eine Concurant immer dem Weg mit dem geringsten Ameisenhaufen-Pheromon P_A. Bei Gleichstand wählt sie einen beliebigen Weg.

(b) Wenn eine Concurant die letzte Futtereinheit genommen hat, dann updatet diese keine Futter-Pheromone auf dem Rückweg. Ansonsten nehmen wir an, dass wann immer eine Concurant aus dem Knoten k_1 einen Weg p zum Knoten k_2 genommen hat, aktualisiert diese danach das Futter-Pheromon nach folgender Regelung:

$$P_F(k_2, p^{-1}) = \begin{cases} \min\left\{P_F(k_2, p^{-1}), r\right\} & \text{wenn } \textit{Abenteurer} \\ r & \text{else} \end{cases},$$

wobei r den Abstand zur Futterquelle und p^{-1} der umgekehrte Weg (also von k_2 nach k_1) ist.

Reiner-Theo ist klar, dass diese Taktik zwar nicht die effizienteste ist und nicht der exakten Lebenswirklichkeit der Concurants entspricht. Allerdings sollte sie einfach genug sein, um eine Grundlage für seine Forschung zu bilden.

2 Aufgabenstellung

Aufgabe 1: Klassisches Ameisenverhalten

Implementieren Sie das oben beschriebene Szenario. Wir stellen Ihnen ein Framework zur Verfügung, das insbesondere bereits nötige Datenstrukturen und einen Teil des Algorithmus implementiert. Ihre Aufgabe ist es, das Verhalten der Concurants und die Kommunikation zu implementieren.

Sie sollen dabei jede Concurant als eigenen Thread modellieren. Jeder Thread (nicht nur alle Concurants) muss auf den angezeigten interrupt-Status hin terminieren. Jede Concurant muss sich ihre Sequenz sowie die bereits untersuchten Wege selbst merken.

Kommunikation von Threads findet einzig und allein wie beschrieben statt. Insbesondere sind etwaige nicht spezifizierte (globale und lokale) Kommunikationsstukturen verboten.

In den von uns bereitgestellten Szenarien ist die Anzahl der Ameisen, sowie deren individuelle Geduld u und Tarnfähigkeit t enthalten.

Um die Arbeit Ihres Programmes überprüfen zu können, müssen Ihre Concurants alle Aktionen über das von uns bereitgestellte `Recorder`-Interface melden.

Ihre Implementierung muss sicherstellen, dass die Anzahl der zulässigen Ameisen in einem Punkt also nie überschritten wird und sich niemals mehrere Ameisen in gleicher Richtung auf einem Weg befinden.

Stellen Sie weiterhin sicher, dass es zu keinen Deadlocks kommen kann und keine Data Races auftreten.

Die Methode `Simulator.run()`, die Sie vervollständigen, soll erst terminieren, wenn

- die spezifizierte Futtermenge zum Ameisenhaufen gebracht wurde, oder
- alle Ameisen terminieren, oder
- das spezifizierte Zeitlimit erreicht wurde.

Aufgabe 2: Modernes Ameisenverhalten

Die stellvertretende Leiterin des Zentrum für Bioinformatik, No Hau, ist über die Annahmen von Reiner-Theo entsetzt. Während seine *empirischen Beobachtungen* unbestritten sind, glaubt sie aufgrund des hohen Intelligenzgerades nicht an die Annahmes des klassischen Ameisenverhaltens.

Sie postuliert folgendes alternatives Verhalten der Concurants:

(a) Concurants benutzen bis zu 8 verschiedene Pheromone zur Kommunikation

(b) Concurants können sich auch grundsätzlich *probabilistisch* entscheiden, und nicht nur bei Pheromonen mit gleichen Werten.

Implementieren Sie zusätzlich zum oben beschriebenen Vorgehen eine alternative Strategie, welche No Hau's Postulaten folgt. Versuchen Sie dabei, dafür zu sorgen, dass die Concurants möglichst schnell möglichst viel Futter einsammeln.

Achten Sie aber darauf, dass die Concurants weiterhin korrekt nebenläufig arbeiten und alle empirisch erhobenen Beobachten weiterhin gelten.

Die Bearbeitung dieser Aufgabe ist freiwillig.

3 Formalia

3.1 Gruppenbildung und Anmeldung zum Projekt

Das Projekt wird in Gruppen von drei Personen bearbeitet[3]. Im Normalfall befinden Sie sich bereits in einer Gruppe mit drei Studierenden. Für den Fall, dass Sie die Zusammensetzung Ihrer Gruppe trotzdem ändern möchten, eröffnen wir bis Mittwoch, den 14.07., 23:59 erneut die Möglichkeit, die Gruppen neu zuzuordnen. Bitte finden Sie selbstständig Kommilitoninnen oder Kommilitonen, um Ihre Gruppe zu ergänzen. Wir haben dafür auch einen Post im Forum angelegt. Wenn Ihre Gruppe trotzdem unvollständig ist, wenden Sie sich ebenfalls bis Mittwoch, 14.07. 23:59 an uns.

Sie werden Ihr Projekt in einem von uns bereitgestellten Repository in unserem GitLab bearbeiten. Registrieren Sie sich bitte daher ebenfalls bis Mittwoch, 14.07. 23.59 in unserem GitLab und speichern Sie Ihren Nutzernamen im mCMS auf Ihrer Persönlichen Statusseite.

Wenn Sie sich nicht in unserem GitLab registrieren, oder vergessen uns Ihren Nutzernamen mitzuteilen, können wir Ihr Projekt nicht anlegen und Sie können das Projekt nicht bearbeiten.

In Ihrem Repository liegt eine Datei `TEAMNAME`. In dieser Datei steht ein anonymisierter Teamname, unter dem wir Ihre Ergebnises anzeigen werden. Sie dürfen diesen Namen jederzeit ändern[4]. Wenn Sie durch diese Namensänderung Ihre Anonymität aufheben, liegt das in Ihrer Verantwortung.

3.2 Meilensteine

Die Einreichung Ihrer Lösungen geschieht via Git. Stellen Sie sicher, dass Ihre Implementierung zum Zeitpunkt der Abgabe auf dem `master` Branch liegt. Zur Einreichung von Dokumenten nutzen Sie bitte das Wiki Ihres Repositories. Wir werden keine anderen Dateien in Ihrem Repository als Dokumente annehmen/bewerten.

3.2.1 Meilenstein *A*

Diesen *Meilenstein A* können Sie bis Montag, *26.07.2021 23:59* einreichen. Dieser Meilenstein umfasst ein Dokument ($\leq$ 9000 non-whitespace Zeichen, sowie zusätzliche Diagramme und Zeichnungen), in welchem Sie Ihre Lösungsansatze darlegen. Bitte nennen Sie dieses Dokument im Wiki `Alpha`.

Wir empfehlen eindringlich, diesen ersten Meilenstein einzureichen. Dies ist nicht verpflichtend, Sie können *auf eigenes Risiko* darauf verzichten.

Wenn Ihr Dokument wesentliche Teile auslässt/Fehler aufzeigt/Fragen offen lässt, bietet ihnen Ihr Tutor in der selben Woche einen Individualtermin an, in dem er die Probleme mit Ihnen bespricht.

Sie bestehen Meilenstein *A* wenn Sie

- Ein Dokument ($\leq$ 9000 non-whitespace Zeichen sowie zusätzliche Diagramme und Zeichnungen) einreichen, welches eine hinreichende Beschäftigung mit der Aufgabenstellung dokumentiert.
- Insofern nötig, den Individualtermin mit Ihrem Tutor wahrnehmen.

3.2.2 Meilenstein Ω

Dieser *Meilenstein* Ω stellt das Ende des Projektes dar. Sie müssen Ω bis Freitag, *20.08.2021 23:59* einreichen.

Ihre Lösung muss die folgenden Inhalte umfassen:

- Ihre Implementierung, die in Form von mit Gradle baubarem *Java-16*- oder alternativ *Go*-Code verfasst ist. Achten Sie unbedingt darauf, dass Ihr Code angemessen kommentiert und insgesamt selbsterklärend ist.
- Ein Dokument ($\leq$ 9000 non-whitespace Zeichen und zusätzliche Diagramme und Zeichnungen), in welchem Sie Ihren Lösungsansatz darlegen. Bitte nennen Sie das Dokument im Wiki `Omega`.

  Erklären Sie dabei, wie genau Ihre Implementierung das korrekte nebenläufe Verhalten der Concurants sicherstellt. Gehen Sie auch darauf ein, wie Ihre Implementierung Deadlocks verhindert.

[3] Es wird maximal eine Ausnahme von dieser Regel geben.
[4] Insofern Ihr Name nicht gegen offensichtliche Regeln unserer Gesellschaft verstößt.

Wir werden Gruppen, die das praktische Projekt (Meilenstein Ω) nicht bestehen sollten, eine Möglichkeit zur Nacharbeit einräumen, wenn die folgenden Bedingungen erfüllt sind:

- Meilenstein A wurde bestanden.
- Meilenstein Ω wurde in einer Form eingereicht, die eine hinreichende Beschäftigung mit der Aufgabenstellung dokumentiert.

Gruppen, welche das praktische Projekt (Meilenstein Ω) nicht bestehen sollten und die gerade genannten Bedingungen nicht erfüllen, bekommen keine Möglichkeit zur Nacharbeit, sondern haben die Vorlesung nicht bestanden.

3.3 Abnahme

Nach Abgabe des Meilensteins Ω und der anschließenden Korrektur müssen Sie schlussendlich Ihr Projekt in einer kurzen Abnahme verteidigen. Im Regelfall werden die Abnahmen am 03.09.2021 stattfinden; Sie können aber bei Bedarf einen anderen Termin mit uns vereinbaren.

3.4 (Automatisiertes) Testen

Wir werden Ihnen Tests für Ihr Projekt zur Verfügung stellen. Es gibt sowohl Tests, die Sie selbst ausführen können, als auch Tests, welche serverseitig automatisiert ausgeführt werden. Wir stellen diese nach bestem Wissen und Gewissen zur Verfügung und können deren Richtigkeit sowie Verfügbarkeit nicht garantieren.

Rufen Sie sich in Erinnerung, dass ein fehlgeschlagener Test zwar auf ein Problem in Ihrem Programm (oder in unserem Test) hindeutet, aber ein bestandener Test nicht die Abwesenheit von solchen Problemen indiziert.

Es ist explizit erlaubt, selbstgeschriebene Tests Ihren Komilitonninen und Komilitonen zur Verfügung zu stellen. Nutzen Sie dafür gerne das Forum.

3.5 Framework

Wir stellen Ihnen ein Framework in Java bereit, das Sie als Basis für Ihre Implementierung benutzen sollen und das gleichzeitig als genaue Spezifikation der Eingabe- und Ausgabeformate dient. Diese Implementierung wird Ihnen in Ihrem persönlichen Repository *voraussichtlich* im Laufe des Donnerstag, 15.07.21, zur Verfügung gestellt.

Mit `git pull` können Sie dann später eventuelle Änderungen an unserer sequentiellen Implementierung in Ihr Projekt übernehmen.

Sie dürfen die im Framework als unveränderlich gekennzeichneten Klassen, Methoden und Signaturen nicht verändern.

Unser Framework enthält Javadoc-Kommentare, die Sie mit Gradle bauen können. Gradle dient auch zum Ausführen der enthaltenen Tests.

3.6 Wahl der Programmiersprache

Im Regelfall werden Sie das Projekt in Java bearbeiten.

Alternativ können Sie das Projekt wahlweise in Go bearbeiten. Für diese Option bieten wir leider kein Framework an. Falls Sie Go nutzen, muss Ihr Code vollständig kompatibel zu unserem Java-Framework sein. Insbesondere müssen Sie die gleichen Dateien lesen können, der Aufgabenstellung folgen und Ihre Implementierung muss das gleiche Verhalten zeigen wie unser Java Framework.

3.7 Bestehenskriterien

Um das Projekt bestehen zu können, müssen Sie die folgenden Punkte erfüllen:

- Ihr Programm muss Aufgabe 1 korrekt lösen.
- Ihr Programm muss (hinreichend effizient) nebenläufig arbeiten.
- Sie müssen uns davon überzeugen, dass Sie das Projekt gemeinsam im Team bearbeitet haben.
- Sie müssen argumentieren können, warum Ihr Programm garantiert korrekt arbeitet.

Um den letzten Punkt zu erfüllen können Sie nach Ihrer Wahl die Aussagen aus der Vorlesung verwenden oder mit der Java Language Specification in Version 16 argumentieren. [5]

[5] Es wird nicht ausreichen, darauf zu verweisen, dass die verwendeten Strukturen laut Spezifikation *thread-safe* sind, da es sich dabei um ein Versprechen ohne präzise festgelegte Bedeutung handelt. Außerdem können Sie sich *nicht* auf die konkrete Implementierung der von Ihnen verwendeten Standard-Java-Klassen berufen, da Ihre Implementierung auf *jeder* (korrekten) Java-Runtime korrekt arbeiten muss.

3.8 Bonus

Sofern Sie einen Bonus (von 0,3 bzw. 0,4 auf die Endnote des gesamten Moduls) erhalten wollen, müssen Sie neben Aufgabe 1 auch Aufgabe 2 korrekt implementieren. Kriterien, die für die Bestimmung des Bonus Berücksichtigung finden, sind

- effektive Parallelisierung,
- effektives Verhalten Ihrer Concurants,
- effiziente und elegante Implementierung, und
- vorbildliche Dokumentation.

Ohne der finalen Evaluation der eingereichten Lösungen vorzugreifen, rechnen wir mit etwa 3-5 Gruppen, die einen Bonus erhalten werden.

4 Checkliste

- ☐ Mittwoch, 14.07., 23:59:
  - ☐ Team Einteilung
  - ☐ Registrierung im GitLab
  - ☐ Nutzernamen im mCMS eintragen.
- ☐ Montag, 26.07., 23:59: Abgabe Meilenstein A
- ☐ Freitag, 20.08., 23:59: Abgabe Meilenstein Ω
- ☐ Freitag, 03.09.: Projektabnahme