1 Einleitung

TODO

2 Pakete und Klassen

2.1 Paket `edu.kit.loop.model`

Das Modell beinhaltet Klassen und Methoden zum Starten und Abbrechen von Simulationen, sowie zum Erstellen und Speichern von Konfigurationen, Stufenspielen, Strategien und Populationen.

2.1.1 Class `UserConfiguration`

//vielleicht im falschen Package

Diese Klasse repräsentiert eine vom Nutzer erstellte Konfiguration. Sie bietet Methoden zum Lesen aller zugehörigen Parameter.

Konstruktoren:

Methoden:

2.1.2 Class `Configuration`

Diese Klasse repräsentiert die elementare Konfiguration einer einzelnen Wiederholung und enthält alle Informationen zum Start einer solchen:

- Stufenspiel
- Anzahl von Agenten
- Runden pro Wiederholung
- Ob gemischte Strategien zugelassen sind
- Gruppen-/Segmenteinteilungen
- Kapital- und Strategieinitialisierung der Segmente

- Agentenpaarung
- Erfolgsquantifizierung
- Adaptionsmechanismus
- Gleichgewichtskriterium
- Maximale Zahl von Adaptionsschritten

Konstruktoren:

Methoden:

2.2 Paket `edu.kit.loop.model.simulator`

Dieses Paket enthält das Interface `Simulator`. Dieses bietet eine Schnittstelle zum Starten und Abbrechen von Simulationen. Beim Start einer Simulation wird eine Referenz auf ein `Simulation`-Objekt zurückgegeben, über das der Ausführungsstatus und die Ergebnisse der Simulation abgefragt werden können.

2.2.1 Class `Simulation`

Ein `Simulation`-Objekt enthält Informationen zu einer gestarteten Simulation, etwa deren Konfiguration, Ausführungsstatus, `id` und gegebenenfalls die Ergebnisse der Simulation. Es wird von einem `Simulator` erzeugt und bereitgestellt, wenn eine Simulation gestartet wird.

Die Klasse unterscheidet nicht zwischen Multikonfigurationen und Nicht-Multikonfigurationen, geht also allgemein von mehreren zugrundeliegenden elementaren Konfigurationen aus.

Konstruktoren:

- `Simulator(UserConfiguration config, int id)`

  Erzeugt ein neues `Simulation`-Objekt mit der gegebenen `UserConfiguration` und der gegebenen `id`.

  `config`: die Konfiguration der Simulation

  `id`: die `id` dieser Simulation

Methoden:

- `protected void addIterationResult(IterationResult result, int i)`

Fügt ein Wiederholungsergebnis zur `i`-ten elementaren Konfiguration dieser Simulation als `IterationResult` hinzu.

`result`: das Wiederholungsergebnis, das hinzugefügt werden soll

`i`: die elementare Konfiguration, zu der das Wiederholungsergebnis hinzugefügt werden soll

- `void registerIterationFinished(Consumer<IterationResult> action)`

  Registriert eine Aktion, die jedes mal ausgeführt wird, wenn eine Wiederholung dieser Simulation abgeschlossen wird. Der Aktion wird als Argument das Ergebnis der Wiederholung als `IterationResult` übergeben.

  `action`: Die Aktion, die bei Abschluss jeder Wiederholung dieser Simulation ausgeführt werden soll

- `List<IterationResult> getIterationResults(int i)`

  Gibt eine Liste der Wiederholungsergebnisse der bisher abgeschlossenen Wiederholungen mit der `i`-ten zugrundeliegenden elementaren Konfiguration als `IterationResult`s zurück.

  `i`: die elementare Konfiguration, zu der die Wiederholungsergebnisse zurückgegeben werden sollen

  Returns: die Wiederholungsergebnisse der bisher abgeschlossenen Wiederholungen mit der `i`-ten zugrundeliegenden elementaren Konfiguration

- `UserConfiguration getUserConfiguration()`

  Gibt die `UserConfiguration` dieser Simulation zurück.

  Returns: die `UserConfiguration` dieser Simulation

- `int getConfigurationCount()`

  Gibt die Zahl der dieser Simulation zugrundeliegenden elementaren Konfigurationen zurück. Im Falle einer Multikonfiguration also die Anzahl verschiedener Werte des Multikonfigurationsparameters, ansonsten 1.

  Returns: die Zahl der dieser Simulation zugrundeliegenden elementaren Konfigurationen

- `int getId()`

  Gibt die `id` dieser Simulation zurück.

  Returns: die `id` dieser Simulation

2.2.2 interface `Simulator`

Über einen Simulator können Simulationen gestartet und abgebrochen werden. Zum Starten einer Simulation muss dem `Simulator` eine `UserConfiguration` übergeben werden,

die die durchzuführende Simulation spezifiziert. Daraufhin wird ein `Simulation`-Objekt erzeugt und zurückgegeben, über das der Ausführungsstatus und die Ergebnisse der gestarteten Simulation abgefragt werden können. Jeder Simulation wird beim Start eine eindeutige `id` zugewiesen.

Methoden:

- `Simulation startSimulation(UserConfiguration config)`

  Startet eine Simulation mit der gegebenen `UserConfiguration` und gibt ein `Simulation`-Objekt zu der gestarteten Simulation zurück.

  `config`: die Konfiguration, mit der die Simulation ausgeführt werden soll

  Returns: ein `Simulation`-Objekt zu der gestarteten Simulation

- `Simulation startSimulation(UserConfiguration config, Consumer<Simulation> action)`

  Startet eine Simulation mit der gegebenen Konfiguration und gibt ein `Simulation`-Objekt zu der gestarteten Simulation zurück. Führt die gegebene Aktion mit der zurückgegebenen `Simulation` als Argument aus, sobald die Ausführung der Simulation abgeschlossen ist.

  `config`: die Konfiguration, mit der die Simulation ausgeführt werden soll

  `action`: Die Aktion, die ausgeführt werden soll, sobald die Simulation abgeschlossen ist

  Returns: ein `Simulation`-Objekt zu der gestarteten Simulation

- `boolean stopSimulation(Simulation sim)`

  Falls die gegebene Simulation aktuell läuft, wird diese abgebrochen und `true` zurückgegeben. Andernfalls wird `false` zurückgegeben.

  `sim`: Die Simulation, die abgebrochen werden soll

  Returns: `true`, wenn die Simulation erfolgreich abgebrochen wurde, `false` sonst

- `boolean stopSimulation(int id)`

  Falls eine Simulation mit der gegebenen `id` läuft, wird diese abgebrochen und `true` zurückgegeben. Andernfalls wird `false` zurückgegeben.

  `id`: Die `id` der Simulation, die abgebrochen werden soll

  Returns: `true`, wenn die Simulation erfolgreich abgebrochen wurde, `false` sonst

- `void stopAllSimulation()`

  Bricht die Ausführung aller gestarteten Simulationen ab.

- `Simulation getSimulation(int id)`

  Gibt das `Simulation`-Objekt der Simulation mit der entsprechenden `id` zurück, falls existent. Ansonsten `null`.

  `id`: Die `id` der Simulation, deren `Simulation`-Objekt zurückgegeben werden soll

Returns: das `Simulation`-Objekt der Simulation mit der entsprechenden `id`, falls existent. Ansonsten `null`.

2.2.3 Class `ThreadPoolSimulator`

Implements: `Simulator`

Eine Implementierung des `Simulator`-Interfaces. Führt die Wiederholungen parallel in einem `ThreadPool` aus.

Konstruktoren:

- `ThreadPoolSimulator()`
  Erzeugt einen neuen `ThreadPoolSimulator`.
- `ThreadPoolSimulator(int maxThreads)`
  Erzeugt einen neuen `ThreadPoolSimulator` mit der gegebenen maximalen Anzahl von `Thread`s.
  `maxThreads`: die maximale Anzahl von `Thread`s im `ThreadPool`

Methoden:

- `int getRunningIterationCount()`
  Gibt die Zahl aktuell ausgeführter Wiederholungen zurück.
  Returns: die Zahl aktuell ausgeführter Wiederholungen
- `int getQueuedIterationCount()`
  Gibt die Zahl aktuell auf Ausführung wartender Wiederholungen zurück.
  Returns: die Zahl aktuell auf Ausführung wartender Wiederholungen

2.2.4 Class `ConfigurationCreator`

Diese Klasse nimmt eine `UserConfiguration` entgegen und extrahiert daraus alle zugehörigen elementaren Konfigurationen. Diese werden als `Configuration`s zurückgegeben.

Konstruktoren:

- `ConfigurationCreator()`

Erzeugt einen neuen `ConfigurationCreator`.

Methoden:

- `List<Configuration> generateConfigurations(UserConfiguration config)`

  Erzeugt alle zu der gegebenen `UserConfiguration` gehörigen elementaren Konfigurationen und gibt sie als `Configuration`s zurück.

  `config`: Die `UserConfiguration`, zu der alle elementaren Konfigurationen generiert werden sollen

  Returns: alle zu der gegebenen `UserConfiguration` gehörigen elementaren Konfigurationen als `Configuration`s

2.3 Paket `edu.kit.loop.model.simulationengine`

3 Glossar

Elementare Konfiguration: Eine Konfiguration, in der Multikonfiguration deaktiviert ist. Mit den „einer Konfiguration zugehörigen elementaren Konfigurationen“ wird im Falle einer Multikonfiguration die Menge aller elementaren Konfigurationen bezeichnet, in denen der Multikonfigurationsparameter die festgelegte Wertemenge durchläuft. Im Falle einer elementare Konfiguration ist wieder die Konfiguration selbst gemeint.