252-0027-00: Einführung in die Programmierung

Übungsblatt 8 (Bonus)

Abgabe: 19 November 2023, 19:00

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Importieren Sie das Eclipse-Projekt für die Bonusaufgabe.

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

Aufgabe 1: Game (Bonus!)

Sie implementieren in dieser Aufgabe Teile eines einfachen Spiels, in dem Teilnehmer verschiedene Aktionen durchführen können.

Die Klasse `Game` repräsentiert ein Spiel und verwaltet die Teilnehmer (jeder Teilnehmer kann nur zu einem Spiel gehören und höchstens 100 Teilnehmer können an einem Spiel teilnehmen). Alle Teilnehmer sind `Humans` und haben als Attribute einen Gesundsheitslevel (`health`) und eine Position (`position`); beides sind ganze Zahlen (`int`). Ein `Human` ist *alive*, wenn `health` > 0.

Teilnehmer planen mittels `Human.scheduleAction(Action)` eine Aktion. Eine Aktion kann den Zustand des `Humans` verändern, für den `scheduleAction` aufgerufen wird (diesen `Human` nennen wir die *Source*), oder die Aktion verändert den Zustand anderer `Humans` des Spiels. Die zulässigen Aktionen sind `Action.ATTACK` und `Action.SUMMON`. Aktionen werden entweder am Ende der aktuellen Runde ausgeführt (`delay` ist 0), oder die Aktionen werden ausgeführt, nachdem `delay` weitere Runden gespielt wurden. Alle Aktionen (am Ende einer Runde) werden immer in der Reihenfolge ausgeführt, in der sie durch `scheduleAction` geplant wurden. Eine Aktion wird nur dann ausgeführt, wenn die *Source* zum Zeitpunkt der Ausführung noch immer *alive* ist. Dabei werden `health` und `position` zum Zeitpunkt der Ausführung verwendet.

In der Welt des Spiels gibt es 1000 Positionen (0, 1, ..., 999). Auf jeder Position können beliebig viele Teilnehmer Platz nehmen.

Für jeden Teilnehmer muss `getPosition()` die aktuelle Position und `getHealth()` den aktuellen Gesundsheitslevel liefern. Die Position wird nur als Folge einer Aktion (siehe weiter unten) geändert. Wenn ein Teilnehmer n Einheiten Schaden erleidet, dann wird `health` um n reduziert. Sie können davon ausgehen, dass jeder Teilnehmer mit `health` > 0 im Spiel anfängt. Ein Teilnehmer, der nicht mehr *alive* ist, kann keine weiteren Aktionen durchführen **und** Aktionen, die er geplant hat, aber die noch nicht ausgeführt wurden, werden *nicht* mehr ausgeführt.

Es gibt drei Arten von Teilnehmern: `Jester`, `Warrior` und `Cleric`. Die folgende Tabelle zeigt den `delay`, der bei den möglichen Aktionen (`ATTACK` und `SUMMON`) für die drei Arten von Teilnehmern angewendet wird.

Source	ATTACK	SUMMON
Jester	0	0
Warrior	0	1
Cleric	0	2

Die nächste Tabelle gibt an, welchen Effekt die Aktionen haben. Beachten Sie, dass für `health` und `position` der aktuelle Wert zur Zeit der Ausführung massgebend ist. Für Task (a) werden Sie `Jester` und `Warrior` implementieren und für Task (b) werden Sie zusätzlich `Cleric` implementieren.

Source	ATTACK	SUMMON
Jester	nichts	nichts
Warrior	Wenn die Source in Position i ist, dann sinkt der Gesundheitslevel aller Teilnehmer in der Nachbarschaft (also mit Position j mit $j \in \{i+1, i-1\}$) um 10 Einheiten.	(Delay $= 1$) Der Gesundheitslevel der Source sinkt um 5 Einheiten.
Cleric	Wenn die Source in Position i ist, dann sinkt der Gesundheitslevel aller Teilnehmer in der Nachbarschaft (also mit Position j mit $j \in \{i+1, i-1\}$) um 3 Einheiten.	(Delay $= 2$) Wenn die Source in Position i ist, dann werden alle Teilnehmer, die *alive* sind und sich in Position j mit $j \in \{i+3, i+4, i+5, i-3, i-4, i-5\}$ befinden, nach Position i bewegt.

Die Teilnehmer haben die Methode `scheduleAction(action)`, die das oben beschriebene Verhalten veranlassen soll (wobei die Aktion am Ende einer Runde ausgeführt wird). Sollte ein Teilnehmer, der nicht mehr *alive* ist, eine Aktion planen, so soll `false` zurückgegeben werden. Sollte ein Teilnehmer X eine Aktion planen, wenn es noch eine Aktion gibt, welche den Teilnehmer X als Source hat und noch nicht ausgeführt wurde, dann soll ebenfalls `false` zurückgegeben werden. Anderenfalls soll `scheduleAction(action)` das beschriebene Verhalten veranlassen und `true` zurückgeben.

Die Methode `Game.advanceTurn()` beendet eine Runde und führt, wie in der Tabelle beschrieben, die Aktionen aus, die für diese Runde geplant sind. Anschliessend beginnt die nächste Runde.

Die Klasse `Game` hat zusätzlich Methoden, die Teilnehmer zum Spiel hinzufügen: `Game.createJester(health,position)`, `Game.createWarrior(health, position)`, und `Game.createCleric(health, position)` erstellen und registrieren den entsprechenden Teilnehmer. Der Rückgabewert (vom Typ `Human`) ist ein Verweis auf die erstellte Instanz. Diese Methoden müssen von Ihnen implementiert werden. Sie können davon ausgehen, dass für `position` immer $5 <$ `position` < 995 gilt.

Beachten Sie, dass Sie die Signaturen der vorgeschriebenen Methoden nicht ändern dürfen. Sie können aber neue Klassen und Enums definieren, neue Konstruktoren, Methoden und Felder erstellen und das Verhalten der exisierenden Methoden anpassen.

Hinweis: Wir empfehlen, diese Aufgaben mit Arrays mit fester Länge zu implementieren und raten davon ab, andere Datenstrukturen zu verwenden oder selber zu implementieren.

(a) Implementieren Sie die Methoden `Game.createJester(health,position)`, `Game.createWarrior(health,position)`, `Human.scheduleAction(action)` und `Game.advanceTurn()`, so dass sich `Human.scheduleAction(action)` und `Game.advanceTurn()` wie oben beschrieben verhalten.

(b) Implementieren Sie die Methode `Game.createCleric(health,position)` und ändern Sie die Methoden `Human.scheduleAction(action)` und `Game.advanceTurn()`, so dass sich `Human.scheduleAction(action)` und `Game.advanceTurn()` wie oben beschrieben verhalten.