Dokumentation: informatiCup 2020 - Pandemie!

Fynn Schulze, Tobias Thie,
Jan Schmolke und Kevin Rohland

27. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Vorstellung

Nachfolgend wird unsere Lösung dokumentiert. Das Team besteht aus Jan Schmolke, Kevin Rohland, Fynn Schulze und Tobias Thie von der Technischen Universität Braunschweig.

1.2 Projektablauf

Wir haben unseren Projektablauf in vier Schritte unterteilt: Bereitstellung der Daten, Analyse des Szenarios, Strategienbildung und Auswertung der Strategien. Die vier Schritte werden im Folgenden einzeln genauer erklärt.

1.2.1 Bereitstellung der Daten

Die erste Schwierigkeit war, die Daten strukturiert zu untersuchen. Die von dem Server zurückgegebene JSON-Datei lässt sich schlecht verarbeiten. Zuerst haben wir ein ER-Diagramm erstellt, um damit die Tabellenstruktur für eine PostgreSQL-Datenbank zu etabilieren. Hier konnten dann die Daten der Events, Städte, Pathogene und Spiele eingetragen werden. Wir hatten anfangs 100 Spiele mit einem simplen EndRound-Return durchlaufen lassen, um die Spieleigenschaften und verschiedenen Pathogene zu analysieren. Schnell fiel uns auf, dass es zu wenig Daten waren. Also erhöhten wir die Spielanzahl auf 1024.

Für das Einpflegen der Daten nutzten wir ein Python Script, welches die JSON-Dateien, die vom Server kommen, in die PostgreSQL-Datenbank einpflegt. Den Großteil der Daten konnten wir gut in einem relationalen Schema abbilden. Ein Teil der Event-Daten war jedoch zu unterschiedlich, sodass wir diese Daten auch als JSON in eine Spalte geschrieben haben. Da hierauf auch direkt mit PostgreSQL zugegriffen werden kann, führte dies zu keinen weiteren Schwierigkeiten.

1.2.2 Analyse des Szenarios

Da nun die Daten strukturiert verfügbar waren, konnten wir mit der Analyse der Daten beginnen. Es wurde zu Anfang eine Liste der möglichen Pathogene erstellt. Diese wurden später aber nicht fest in die Lösung eingearbeitet, sondern waren nur wichtig, um in der Datenanalyse Unterschiede zwischen den Werten feststellen zu können. Anschließend wurden Tests zu den verschiedenen Kausalitäten der Werte durchgeführt, wie z.B. das Verhältnis der unterschiedlichen Stadteigenschaften zum Infektionsanteil, zur Infektionsrate und zur Todesrate. Dasselbe wurde auch mit den Vireneigenschaften durchgeführt, auf welche wir uns bei den Implementierungsversuchen konzentriert haben.

Für eine bessere Übersicht über die verschiedenen Szenarien wurde ein Tool entwickelt, welches u.a. die Städte auf einer Karte darstellt. Mit diesem Tool wurden Korrelationen und Kausalitäten zwischen den Geokoordinaten und Flugverbindungen untersucht.

1.2.3 Strategienbildung

Der erste Lösundsverusch war testweise ein Bogo-Algorithmus. Später haben wir uns darauf konzentriert, den tödlichsten Virus mit Hilfe einer Risikobewertungsfunktion herauszufinden und diesen in der ersten Runde direkt unter Quarantäne zu stellen, damit er sich nicht ausbreiten kann. Damit sind die Städte vorerst vor dem tödlichsten Virus geschützt, da sich die Anderen im Spiel befindlichen Viren in der zweiten Runde über fast den gesamten Planeten ausbreiten und keine Stadt von mehr als einem Virus befallen sein kann.

Weitere Strategien werden in den folgenden Kapiteln genauer beschrieben.

1.2.4 Auswertung der Strategien

Nachdem wir mehrere Strategien hatten, haben wir die Strategien über Zufallsspiele laufen lassen und verglichen, welche Strategie wie oft gewonnen hat. Die beste Strategie wurde dann als Lösung genommen.

1.3 Ergänzende Lösungsstrategien

Unsere ursprüngliche Idee war, einen evolutionären Algorithmus darauf zu trainieren, die richtige Gewinnstrategie auszuwählen. Es gibt für verschiedene Ausgangsszenarien besser und schlechter geeignete Algorithmen. Dies wurde aber aus zeitlichen Gründen abgebrochen.

Eine andere Möglichkeit wäre es, eine KI direkt auf das gesamte Spielszenario zu trainieren. Dieses ist allerdings so umfangreich, dass die KI sehr schwer zu trainieren wäre, da es zu viele Inputs und Outputs gibt. Aus diesen Gründen wurde auch der 1. Ansatz mit der Auswahl einer vordefinierten Strategie gewählt.

2 Theoretischer Lösungsansatz

2.1 Analyse

2.1.1 Vorgegebene Strukturen

Während einer Spielinstanz werden laut Aufgabenstellung drei verschiedene Entitätentypen übergeben:

- Pathogene: Pathogene besitzen Namen und die vier Eigenschaften *Infektiosität*, *Mobilität*, *Dauer* und *Letalität*. Diese Werte sind unveränderlich.
- Städte: Städte besitzen die unveränderlichen Werte *Namen*, *Koordinaten* und *Flugverbindungen*. Hierzu kommen die veränderlichen Eigenschaften *Population*, *Stärke der Wirtschaft*, *Stabilität der Regierung*, *Hygienestandards* und *Achtsamkeit der Einwohner*. Desweiteren können Städte selber *Events* (Stadtevents) haben.
- Events: Events sind unterteilt in *globale Events* und *Stadtevents*. Events beschreiben einen Großteil des Zustandes einer Spielrunde.

Die Eigenschaften der Pathogene sowie die Eigenschaften *Stärke der Wirtschaft*, *Stabilität der Regierung*, *Hygienestandards* und *Achtsamkeit der Einwohner* der Städte können fünf aufsteigende, diskreten Werte annehmen („−−“, „−“, „0“, „+“, „++“). Die Population von Städten kann nur ganzzahlige, nicht negative Werte annehmen.

Desweiteren werden folgende Funktionen beschrieben:

- Pathogene erscheinen in der ersten Runde eines Spiels.
- Städte können von Pathogenen infiziert werden.
- Infizierte Städte verbreiten das entsprechende Pathogen.
- Die infizierte Population einer Stadt kann sterben oder eine Immunität dem entsprechenden Pathogen gegenüber entwickeln.
- Am Anfang des Spiels beträgt das Aktionspunktekonto 40 Punkte, pro Runde erhöht es sich um 20.

2.1.2 Analyse der Eigenschaften und Funktionen

Um einen Überblick über die Verhaltensweisen der einzelnen Entitäten des Spiels zu erhalten wurde eine Datenbank erstellt (siehe 3.1) und die Auswertung von 1024 Spielen übergeben, bei denen je Runde nur mit dem „EndRound“ Befehl geantwortet wurde. Daraufhin wurde das Spiel auf seine Eigenschaften und Funktionen hin analysiert.

Tabelle 1: Auflistung der gefundenen Pathogene

Name	Dauer	Infektiosität	Letalität	Mobilität
Admiral Trips	−	++	++	+
Azmodeus	0	0	0	0
Bonulus eruptus	+	−−	−−	−−
Coccus innocuus	0	0	−−	−−
Endoictus	+	−	−−	0
Hexapox	0	0	0	+
Influenza iutiubensis	−−	++	−−	++
Methanobrevibacter colferi	−−	0	−	++
Moricillus ☠	−	−	+	0
N5-10	0	+	+	++
Neurodermantotitis	0	+	0	0
Phagum vidiianum	+	0	+	0
Plorps	+	0	−	0
Procrastinalgia	++	−	−−	−−
Rhinonitis	−	0	−−	−
Saccharomyces cerevisiae mutans	0	0	−	0
Shanty	0	+	−	−
Xenomonocythemia	0	−	++	−−
Φthisis	0	0	0	0

Pathogeneigenschaften:

Es wurden 19 Pathogene gefunden, welche in verschiedenen Kombinationen auftreten. Um die Eigenschaften dieser direkt abzuleiten wurde untersucht, wie sich Pathogene mit verschiedenen Eigenschaftsgraden verhalten. Hierzu wurde eine Gruppe von 57 Städten mit gleichen Eigenschaften ausgesucht (Wirtschaft: 0, Regierung: 0, Hygiene: 0, Achtsamkeit: 0) und die Spielverläufe der Infektionen in diesen Städten miteinander verglichen. Hierbei ließ sich zunächst feststellen, dass die Werte der Eigenschaften nicht bedeuten, dass sich Pathogene oder Städte mit gleichen Eigenschaften identisch verhalten. So zeigten alle Städte der untersuchten Gruppe trotz identischen Werten leicht unterschiedliches Verhalten. Auch die zwei identischen Pathogene *Azmodeus* und *Φthisis* verhielten sich bei der Infektion von denselben Städten unterschiedlich. Somit erschien es klar, dass die Werte „−−“, „−“, „0“, „+“, und „++“ nicht diskrete Werte sind, sondern Abbildungen von Wertebereichen.

Es ergaben sich folgende Beobachtungen bei der Infektion:

- In Runde 1 werden 2 bis 3 zufällige Städte mit jeweils einem Pathogen infiziert. Hierbei konnten Pathogene auch mehrfach auftreten.
- Eine Stadt kann jeweils nur von einem Pathogen infiziert sein. Die Stadt kann erst mit einem anderen Pathogen infiziert werden, nachdem das vorherige Infektionsevent überstanden wurde.
- Die Prävalenz des Pathogens in infizierten Städten stieg nach dem selben Muster an: Ein fester Prozentsatz (Infektionsrate) der nicht infizierten Population wird jede Runde neu infiziert. Hierbei hängt die Höhe des Prozentsatzes mit der Höhe der Eigenschaft Infektiosität des Pathogens ab (siehe Tabelle 2).
- Bei der Erstinfektion einer Stadt wird die eben genannte Regel zur Infektion zwei mal angewandt.
- Wenn die Prävalenz des Pathogens in einer Stadt unter 5% fällt, endet das Infektionsevent der Stadt.

Tabelle 2: Infektionsraten

Name	**Durchschnittl. Infektionsrate**	**Infektiosität**
Influenza iutiubensis	0,816	++
Admiral Trips	0,765	++
Shanty	0,646	+
N5-10	0,636	+
Neurodermantotitis	0,581	+
Plorps	0,538	o
Rhinonitis	0,538	o
Coccus innocuus	0,515	o
Azmodeus	0,515	o
Phagum vidiianum	0,419	o
Hexapox	0,386	o
Saccharomyces cerevisiae mutans	0,344	o
Φthisis	0,342	o
Methanobrevibacter colferi	0,286	o
Moricillus ☠	0,216	−
Procrastinalgia	0,216	−
Endoictus	0,172	−
Xenomonocythemia	0,086	−
Bonulus eruptus	< 0,026 *	−−

(* Schätzung. Nach Infektion liegt Prävalenz unter 0,05, was zum sofortigen Abbruch des Infektionsevents führt. Somit konnte kein Wert für *Bonulus eruptus* ermittelt werden)

Folgendes konnten wir beim Infektionsverlauf feststellen:

- Jedes Pathogen hat eine feste Inkubationszeit, abgeleitet von der Eigenschaft Dauer, welche unabhängig von den Eigenschaften der infizierten Städte ist (siehe Tabelle 3)
- Die Inkubationszeit ist die Anzahl an Runden nach der für eine infizierte Population entschieden wird, ob sie stirbt oder Immunität entwickelt.
- Die Höhe des Prozentsatzes (Letalitätsrate) der sterbenden Population wird von der Eigenschaft Letalität abgeleitet (siehe Tabelle 4)
- Die maximale Rundenanzahl, die ein Infektionsevent in einer Stadt aktiv bleibt, ist somit von der Infektionsrate und der Inkubationszeit abhängig (siehe Tabelle 5). Im Trend wird die Rundenanzahl von Höhe der Dauer und invertierter Höhe der Infektiosität abgeleitet.

Tabelle 3: Inkubationszeiten

Name	**Inkubationszeit (Runden)**	**Dauer**
Procrastinalgia	15	++
Phagum vidiianum	10	+
Endoictus	9	+
Plorps	8	+
Neurodermantotitis	7	o
Φthisis	7	o
Coccus innocuus	6	o
Hexapox	6	o
Azmodeus	5	o
Shanty	5	o
N5-10	4	o
Saccharomyces cerevisiae mutans	4	o
Xenomonocythemia	4	o
Moricillus ☠	3	−
Rhinonitis	3	−
Admiral Trips	2	−
Influenza iutiubensis	1	−−
Methanobrevibacter colferi	1	−−

Für *Bonulus eruptus* konnte kein Wert ermittelt werden.

Tabelle 4: Letalitätsraten

Name	**Durschnittl. Letalitätsrate**	**Letalität**
Xenomonocythemia	0,665	++
Admiral Trips	0,611	++
Moricillus ☠	0,538	+
Phagum vidiianum	0,506	+
N5-10	0,474	+
Azmodeus	0,337	o
Hexapox	0,337	o
Neurodermantotitis	0,271	o
Φthisis	0,234	o
Plorps	0,167	−
Saccharomyces cerevisiae mutans	0,132	−
Shanty	0,066	−
Methanobrevibacter colferi	0,064	−
Endoictus	0,045	−−
Rhinonitis	0,032	−−
Influenza iutiubensis	0,031	−−
Procrastinalgia	0,007	−−
Coccus innocuus	0,0002	−−

Für *Bonulus eruptus* konnte kein Wert ermittelt werden.

Tabelle 5: Maximale Infektionsdauer

Name	Max. Infektionsdauer (Runden)	Dauer	Infektiosität
Xenomonocythemia	37	o	—
Procrastinalgia	28	++	—
Endoictus	24	+	—
Phagum vidiianum	17	+	o
Moricillus ☠	16	—	—
Φthisis	15	o	o
Hexapox	13	o	o
Plorps	12	+	o
Saccharomyces cerevisiae mutans	12	o	o
Azmodeus	11	o	o
Neurodermantotitis	11	o	+
Coccus innocuus	10	o	o
N5-10	8	o	+
Shanty	8	o	+
Rhinonitis	7	—	o
Admiral Trips	5	—	++
Influenza iutiubensis	1	——	++
Methanobrevibacter colferi	1	——	o
Bonulus eruptus	0	+	——

Folgendes konnten wir bei der Mobilität feststellen:

- Im Durchschnitt war jede Stadt ab Runde 2 eines Spiels mit einem Pathogen infiziert. Eine Ausnahme bildeten Städte mit einer Population im geringen einstelligen Bereich (z.B. *Vatican City*). Dies erklärt sich dadurch, dass Populationseinheiten nicht prozentual infiziert sein können. Durch Rundung führt dann eine Infektion mit geringer Infektiosität zu einer Infektionsrate von 0 anstatt > 0,05.
- Die mögliche Infizierung der Städte scheint unabhängig vom eigenen Standort, Standort der zuvor infizierten Stadt und der Mobilität des Pathogens zu sein.
- Städte mit Flugverbindungen zu infizierten Städten neigen eher dazu dessen Infektion anzunehmen als andere.

Tabelle 6: Durchschnittliche Anzahl infizierter Städte in Runde 2

Pathogen	Durschn. Anzahl inf. Städte in Runde 2	Mobilität
N5-10	158	++
Hexapox	150	+
Influenza iutiubensis	144	++
Methanobrevibacter colferi	134	++
Admiral Trips	129	+
Neurodermantotitis	128	o
Saccharomyces cerevisiae mutans	119	o
Moricillus ☠	107	o
Azmodeus	105	o
Plorps	92	o
Φthisis	70	o
Phagum vidiianum	69	o
Endoictus	55	o
Rhinonitis	36	−
Shanty	25	−
Coccus innocuus	4	−−
Procrastinalgia	1	−−
Xenomonocythemia	1	−−
Bonulus eruptus	0	−−

Stadteigenschaften:

Die Analyse der Einflüsse der verschiedenen Eigenschaften der Städte erwies sich als komplizierter, als die für die Pathogene. Ein Problem lag in der Anzahl der Pathogene, da es zu wenige gab, um einen durchschnittlichen Grundwert für den Einfluss der Stadteigenschaften zu finden. Desweiteren waren die Eigenschaften der Städte abhängig voneinander. Wenn in einer Eigenschaft ein hoher Wert vorliegt, ist die Wahrscheinlichkeit hoch, dass auch die anderen Eigenschaftswerte nicht niedrig sind. Es ließ sich aber generell ein Zusammenhang zwischen der Höhe der Werte und der Auswirkung auf Infektions- und Letalitätsrate erkennen (siehe Tabelle 7). Je höher die Summe, desto geringer ist die Infektions- und Letalitätsrate.

Um den Einfluss der einzelnen Eigenschaften genauer zu betrachten wurde ein Grundwert für die verschiedenen Raten gesetzt, indem die durchschnittlichen Infektions- und Letalitätsraten der Städte mit den Eigenschaften Wirtschaft : 0, Regierung: 0, Hygiene: 0 und Achtsamkeit: 0 gewählt wurden. Die Gruppe mit diesen Werten ist zudem die größte Gruppe, wenn man die Städte nach Eigenschaftswerten aufteilt. Ausgehend von diesen Grundraten wurde dann untersucht, wie sie sich verändern, wenn jeweils nur eine Eigenschaft steigt oder sinkt (siehe Tabellen 8 und 9). Hier stellte sich heraus, dass insbesondere die Eigenschaften

Tabelle 7: Einfluss der Stadteigenschaften auf Infektions- und Letalitätsrate

Summe Stadteigenschaften	Infektionsrate	Letalitätsrate
-5	0,75624	0,46189
-4	0,74784	0,42903
-3	0,73997	0,42297
-2	0,72636	0,40623
-1	0,71915	0,39158
0	0,69485	0,33939
1	0,66313	0,30313
2	0,66495	0,28558
3	0,6548	0,2704
4	0,64042	0,24726
5	0,63545	0,23388
6	0,62041	0,20915
7	0,60756	0,19331
8	0,61322	0,20503

(Eigenschaftswerte wie folgt umgerechnet: −− = -2, − = -1, o = 0, + = 1, ++ = 2)

Regierung und Hygiene die Letalitätsrate sowie die Infektionsrate beeinflussen.

Tabelle 8: Auswirkung der Stadteigenschaften auf die Letalitätsrate

Letalitätsrate	Wirtschaft	Regierung	Hygiene	Achtsamkeit	Anzahl Städte
0,34795	o	o	o	o	57
0,38282	−	o	o	o	7
0,30476	+	o	o	o	6
0,41383	o	−	o	o	9
0,26423	o	+	o	o	8
0,40249	o	o	−	o	4
0,24095	o	o	+	o	12
0,36316	o	o	o	−	2
0,32356	o	o	o	+	8

Tabelle 9: Auswirkung der Stadteigenschaften auf die Infektionsrate

Infektionsrate	Wirtschaft	Regierung	Hygiene	Achtsamkeit	Anzahl Städte
0,85297	o	o	o	o	57
0,87879	−	o	o	o	7
0,82953	+	o	o	o	6
0,88171	o	−	o	o	9
0,80337	o	+	o	o	8
0,89153	o	o	−	o	4
0,79393	o	o	+	o	12
0,86365	o	o	o	−	2
0,83318	o	o	o	+	8

Events:

Bei den Events konnten wir 11 Stadtevents und 9 globale Events beobachten. Events fungieren als Flags für das Spiel, sodass es unabhängig von der Vorrunde eine folgende Runde generieren kann. Insbesondere gibt es Events, die vorherige Aktionen (siehe Aufgabenstellung) des Spielers beschreiben:

- Stadtevents
  - quarantine: Resultat des Befehls *putUnderQuarantine*
  - airportClosed: Resultat des Befehls *closeAirport*
  - connectionClosed: Resultat des Befehls *closeConnection*
  - campaignLaunched: Resultat des Befehls *closeConnection*
  - electionsCalled: Resultat des Befehls *callElections*
  - hygienicMeasuresApplied: Resultat des Befehls *applyHygienicMeasures*
  - influenceExerted: Resultat des Befehls *exertInfluence*
  - medicationDeployed: Resultat des Befehls *deployMedication*
  - vaccineDeployed: Resultat des Befehls *deployVaccine*

- globale Events
  - medicationInDevelopment: Resultat des Befehls *developMedication*
  - medicationAvailable: Nach Vollendung der Medizinentwicklung, erlaubt Befehl *deployMedication*
  - vaccineInDevelopment: Resultat des Befehls *developVaccine*
  - vaccineAvailable: Nach Vollendung der Impfstoffentwicklung, erlaubt Befehl *deployVaccine*

Weiterhin gibt es Events, welche das Auftreten von Pathogenen beschreiben:

- Stadtevents
  - outbreak: benennt das Pathogen, dass die Stadt befallen hat und gibt die Prävalenz der Infektion an.
  - bioTerrorism: während des Spiels kann eine nicht infizierte Stadt mit einem zufälligen neuen Pathogen infiziert werden
- globale Events
  - pathogenEncountered: gibt Auskunft über die distinkten Pathogene, die sich bereits im Spiel befunden haben

Zuletzt gibt es noch Events, die zufällig oder nach der Erfüllung bestimmter Voraussetzungen aktiviert werden. Für diese konnten die genauen Regeln zeitbedingt nur teilweise bestimmt werden:

- Stadtevents
  - uprising: Voraussetzung unbekannt. Es wurde vermutet, dass es die Stadteigenschaften (z.B. Regierung) negativ beeinflusst.
  - antiVaccinationism: Voraussetzung unbekannt. Es wurde vermutet, dass es die Effektivität von Impfstoffen negativ beeinflusst.
- globale Events
  - economicCrisis: Voraussetzung unbekannt. Während dieses Event aktiv ist, verschlechtert sich in jeder Runde die Wirtschaftseigenschaft zufälliger Städte.

2.2 Lösungsansatz

2.2.1 Ziel

Als Bewertungskriterien wurden folgende Ziele bestimmt:

1. 50 % der Weltpopulation soll überleben
2. Die Anzahl der benötigten Runden für die Lösung soll möglichst gering sein
3. Sollte es nicht möglich sein, 50 % der Population zu retten, gilt jene Lösung als am besten, welche die höchste Rundenanzahl braucht, bis das Spiel abbricht

Desweiteren soll die Lösung so gestaltet werden, dass jede Runde unabhängig von Wissen über den bisherigen Verlauf des Spiels zielführend bearbeitet werden kann.

2.2.2 Strategie

1. Spielstand mit zwei Pathogenen

Bei dem Auftreten von nur zwei Pathogenen am Spielanfang scheint die richtige Handlungsweise einfach. Wenn man eine infizierte Stadt sofort in Runde 1 unter Quarantäne stellt, und dies bis zum Ende des Infektionsausbruches beibehält, werden fast alle anderen Städte in Runde 2 mit dem anderen Pathogen infiziert sein und keine neuen Städte können ab diesem Zeitpunkt mehr infiziert werden, sofern keine neuen Pathogene mittels bioTerrorism-Events auftauchen. Dies verringert die Anzahl der Runden bis zur letzten Infektion auf zwei. Die Entscheidung, welches Pathogen zunächst unter Quarantäne gestellt werden soll, ist jedoch komplizierter. Hier entwickelten sich drei Strategien:

- DeadlyFirst: Um zu garantieren, dass möglichst viele Menschen vor dem Tod gerettet werden, wird anhand eines Gefährlichkeitsgrades bestimmt, welches Pathogen zu den meisten Todesfällen führen würde, und dieses dann bevorzugt unter Quarantäne gestellt. Der Gefährlichkeitsgrad wurde anhand der Analyse als folgende Funktion bestimmt:

$$\text{Gefährlichkeit} = \text{Letalität} + \text{Mobilität} + \frac{\text{Dauer}}{2} \tag{1}$$

  Nachteil dieser Strategie ist, dass weniger auf die Inkubationszeit der Pathogene reagiert wird. Somit kann die Länge der nötigen Quarantäne nicht bestimmt werden. Dies führt unter gewissen Konstellationen von Pathogenen dazu, dass unnötig Punkte in die Quarantäne investiert werden, die sonst z.B. für die Entwicklung von Medikamenten genutzt werden könnten.

- FastFirst: Hier besteht die Strategie darin, dass das Pathogen mit der geringsten Verweildauer in einer Stadt für wenige Runden unter Quarantäne gestellt wird. Da die anderen Pathogene zu diesem Zeitpunkt bereits alle anderen Städte infiziert haben

und längere Verweildauer besitzen, kann das gewählte Pathogen sich so nicht weiter ausbreiten, bevor es unter die Prävalenzrate von 5 % fällt und ausstirbt. Somit stehen früher als bei deadlyFirst Punkte und Optionen zur Verfügung, um weiter in das Spielgeschehen einzugreifen. Die Verweildauer eines Pathogens berechnet sich wie folgt:

$$\text{Verweildauer} = \text{Dauer} - \text{Infektiosität} \tag{2}$$

Nachteil dieser Strategie ist, dass die Überlebensrate der Menschen keine Priorität ist. Unter Umständen werden dadurch Spiele verloren, die mittels DeadlyFirst noch gewonnen werden könnten.

- SlowFirst: Bei diesem Ansatz wird das Pathogen mit der höchsten Verweildauer unter Quarantäne gestellt. Der Vorteil hierbei liegt darin, dass mit der Verweildauer des Pathogens eine feste obere Grenze für die Anzahl der Runden bis Spielende existiert, sofern keine neuen Pathogene hinzukommen. Da auf die Überlebensrate und das Entwickeln von Impfstoffen oder Medikamenten keine Rücksicht genommen wird, beendet diese Strategie das Spiel am schnellsten. Nachteil ist hierbei, dass wie bei FastFirst Spiele verloren werden können, wenn die Gefährlichkeit der Pathogene ignoriert wird. Außerdem sind alle nutzbaren Punkte bis zum Ende des Spiels in die Quarantäne des Pathogens investiert.

Nach der Durchführung der Strategien wird überprüft, ob sich nun nur noch ein Pathogen im Spiel befindet. Wenn dem so ist, wird folgend dafür ein Medikament entwickelt und an die Städte ausgegeben, die die höchste Anzahl an infizierten Menschen besitzen, um die Verweildauer der Infektion global zu senken. Dies wird getan bis das Spiel beendet ist.

In diesem Stadium besteht die höchste Gefahr, das zufällig durch das „bioTerrorism“ Event ein neues Pathogen entsteht. Um die Infektion auf weitere Städte in diesem Fall zu verhindern, wird ab diesem Zeitpunkt darauf geachtet, einen Vorrat von 40 Punkten bereit zu halten, um ein neu aufkommendes Pathogen sofort und bleibend unter Quarantäne zu stellen.

2. Spielstand mit mehr als zwei Pathogenen

Sollte das Spiel mit drei Pathogenen starten, können die gleichen drei Strategieansätze, wie oben für zwei Pathogene beschrieben, für die Quarantäne einer der am Beginn infizierten Städte verwendet werden. Folgend muss aber mit einer anderen Strategie auf das Vorhandensein zweier oder mehrerer Pathogene eingegangen werden. Hierbei ergaben sich auch unterschiedliche Ansätze:

- Vacc: Um zu verhindern, dass das Pathogen mit der längsten Verweildauer auf weitere Städte übergreifen kann, wenn sich diese von der Ausgangsinfektion erholt haben,

wird dem entgegen der Impfstoff entwickelt und an noch nicht infizierte Städte verteilt. Die Zielstädte werden dabei so ausgewählt, dass die Städte mit der höchsten Summe an Stadteigenschaften Vorrang haben, da dort die Verweildauer des Pathogens am längsten wäre. Dies ist besonders Effektiv gegen Pathogene, die eine längere Verweildauer und größere Mobilität haben, da dort oft Neuinfektionen von Städten komplett verhindert werden können. Wenn sich anschließend nur noch ein Pathogen im Spiel befindet, wird, ähnlich wie oben beschrieben, ein Medikament entwickelt und an die Städte ausgegeben, die die höchste Anzahl an infizierten Menschen besitzen.

- Med: Hier wird stattdessen sofort ein Medikament gegen das Pathogen mit der längsten Verweildauer entwickelt und an Städte mit den meisten infizierten Menschen ausgegeben. Hierdurch wird zum Einen die Gesamtzahl gestorbener Menschen, als auch die Rundenanzahl, bis die letzte Stadt von der Infektion befreit wird, gesenkt. Diese Strategie ist am effektivsten, wenn die Pathogene eine besonders kurze Verweildauer oder eine hohe Letalität besitzen.

  Nach der Durchführung dieser Strategien wird wie bei dem Vorgang für zwei Pathogene verfahren.

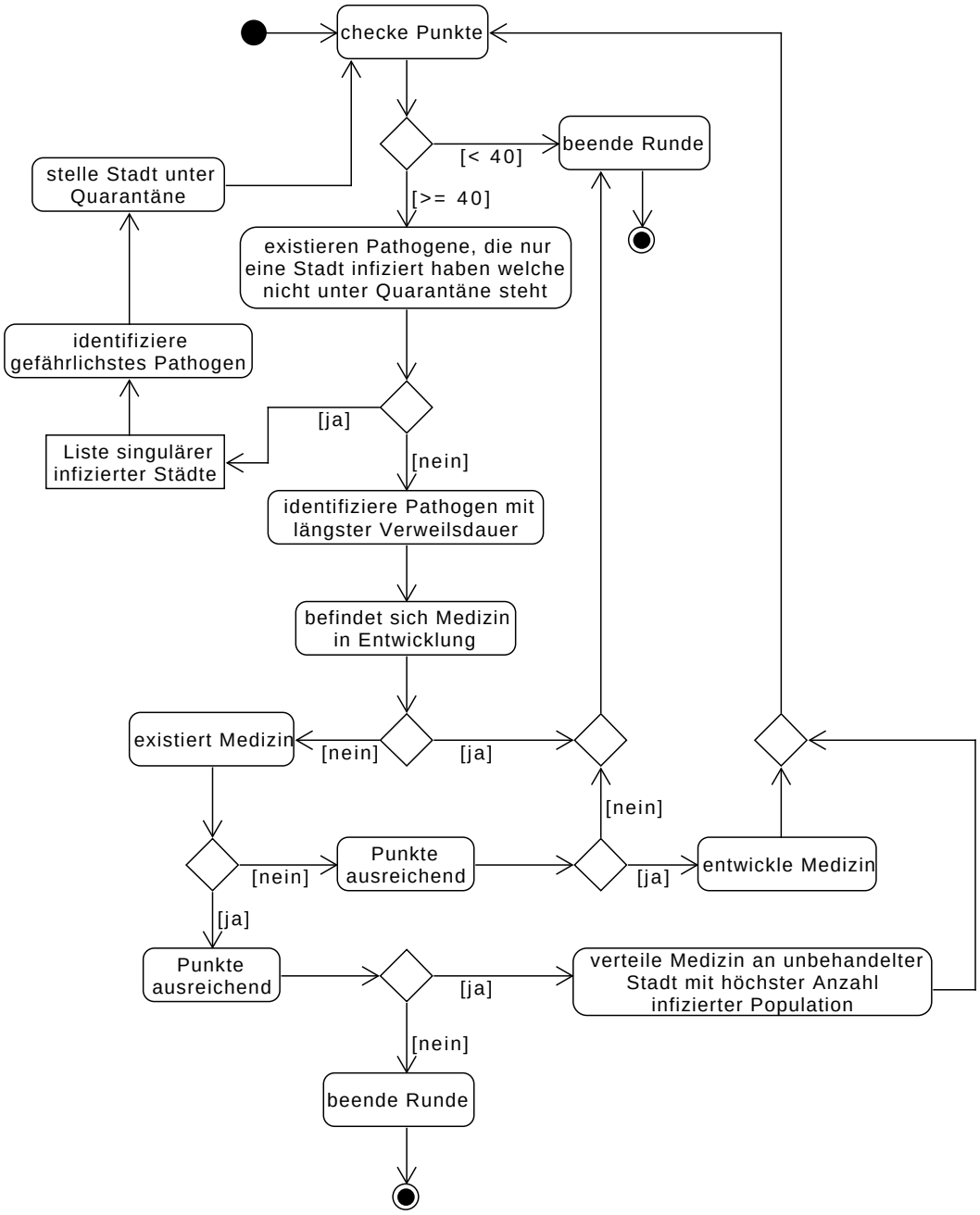


Abbildung 1: Aktivitätsdiagramm der MedDeadlyFirst-Implementation

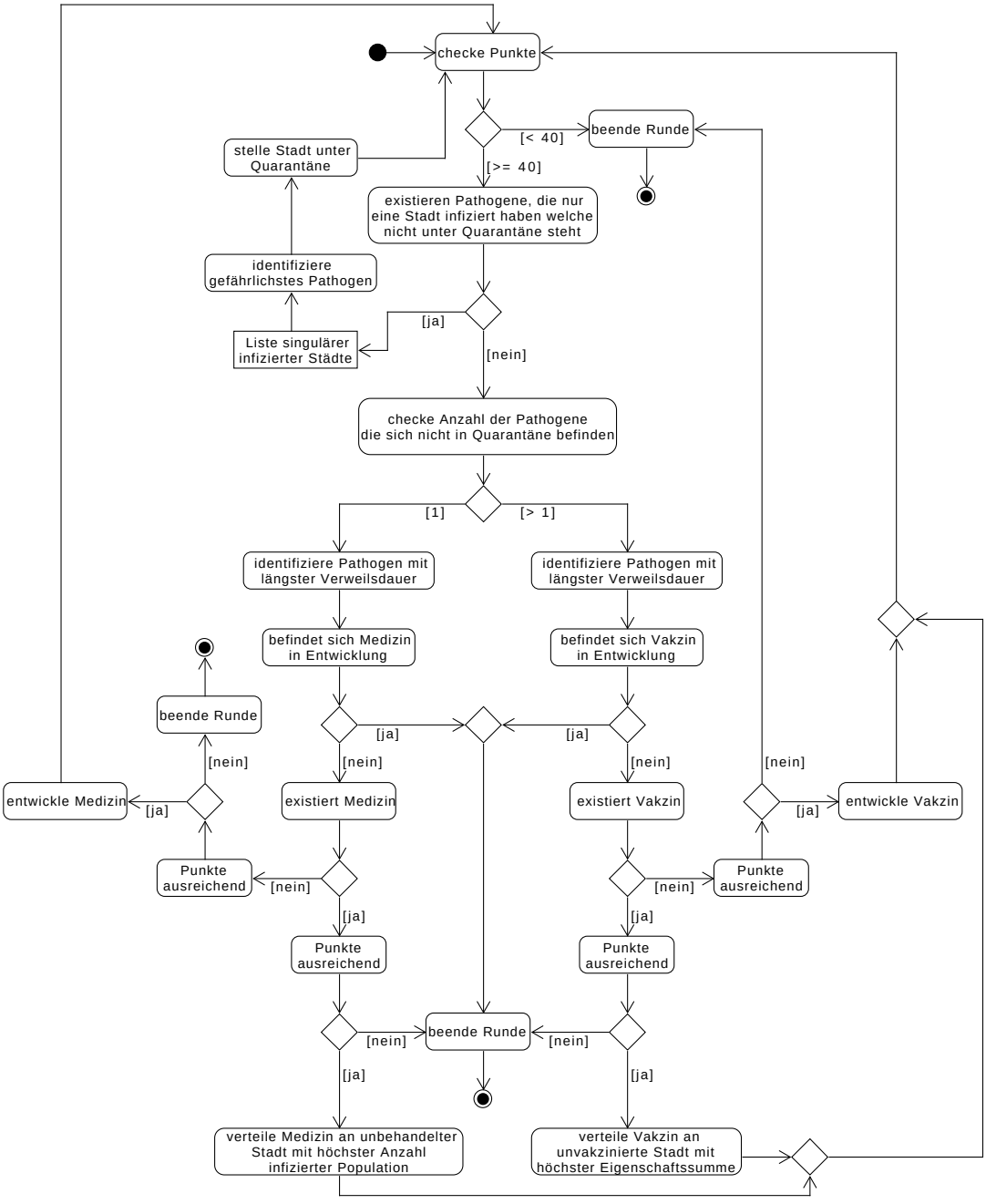


Abbildung 2: Aktivitätsdiagramm der VaccDeadlyFirst-Implementation

Aus den Beschriebenen Strategien wurden dann 6 Kombinationen erstellt, die die oben genannten Lösungswege für zwei oder mehrere Pathogene verwenden:

- VaccDeadlyFirst
- VaccFastFirst
- VaccSlowFirst
- MedDeadlyFirst
- MedFastFirst
- MedSlowFirst

Da sich jede Kombination bei bestimmten Konstellationen von Pathogeneigenschaften als besonders effektiv herausstellte, wurde diskutiert ob sich mit Machine Learning noch eine Funktion erstellen ließe, die für jedes Szenario entscheidet, welche Strategie verwendet werden sollte. Nach einigen Versuchen und verschiedenen Ansätzen konnte jedoch kein gutes Ergebnis erzielt werden, so dass dieser Ansatz verworfen wurde.

3 Softwarearchitektur

Dieses Kapitel beschreibt die Softwarearchitektur von unserem erstellten Webservice, welches dann über eine REST-Schnittstelle mit dem Client kommuniziert. Außerdem wird anfangs noch die Datenmodellierung besprochen, mit welcher die Daten in der Datenbank strukturiert wurden.

3.1 Datenmodellierung

Im Folgenden wird die Datenstruktur der PostgreSQL-Datenbank vorgestellt, welche die Daten für die weitere Analyse besser zugreifbar gemacht hat.

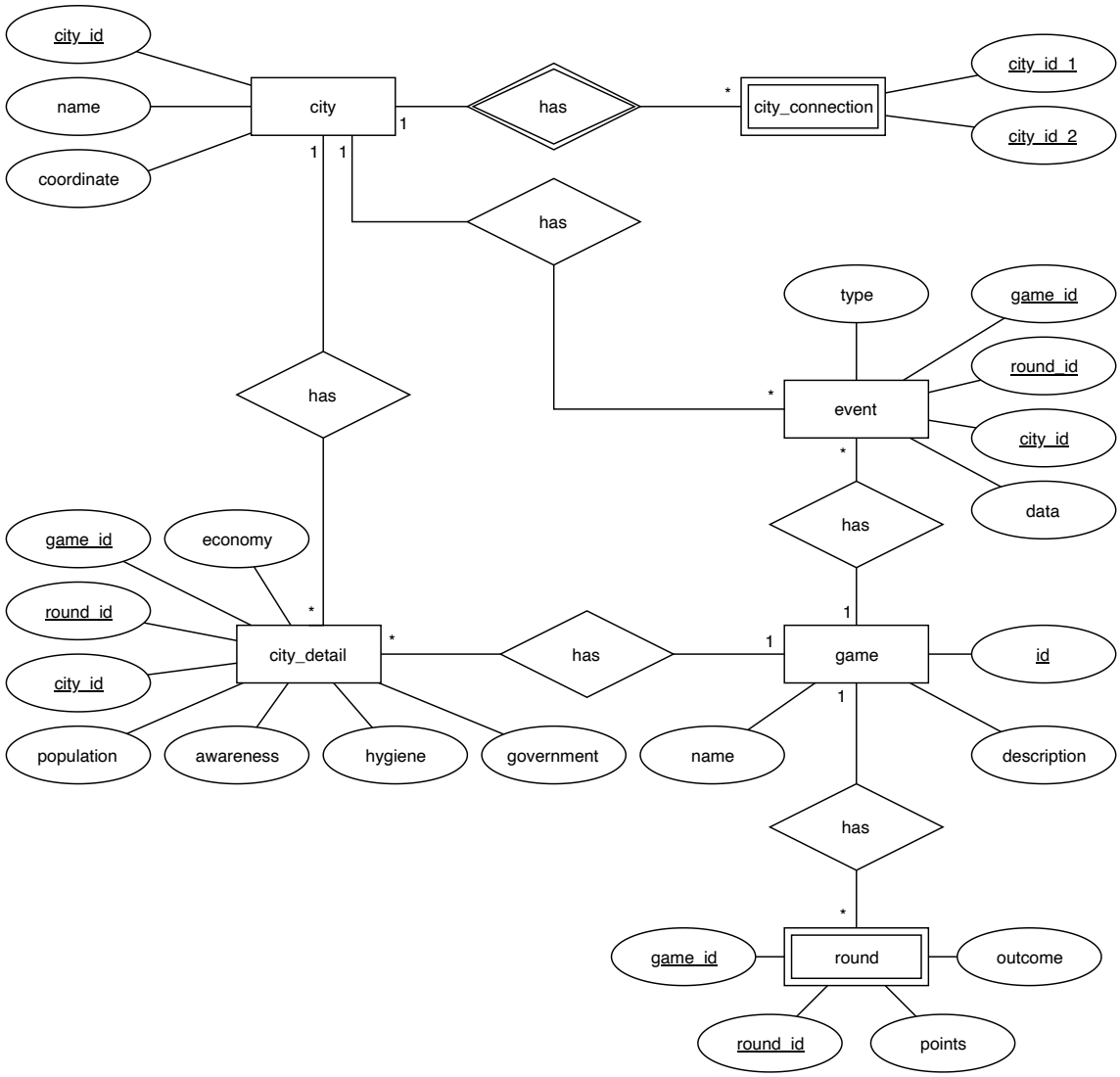


Abbildung 3: ER-Diagramm der Datenbank

Die Datenbank wurde in zwei Arten von Tabellen eingeteilt. Die Tabellen city und city_connection enthalten keine game_id und round_id, da die Städte und die Stadtverbindungen statische Eigenschaften über die verschiedenen Spiele hinweg sind. Die anderen Tabellen (city_detail, event und round) sind über die Tabelle game verknüpft. Für die praktische Analyse war es häufig notwendig city_detail und event miteinander zu joinen. Dies geschah in der Regel über die game_id, round_id und city_id.

3.2 Architektur

Für unseren Webservice nutzen wir das Spring Framework[1], welches ein offenes Framework für Java ist, um Webanwendungen zu schreiben. Unsere Software startet Spring und registriert einige Controller, womit wir bei Spring das URL-Mapping festlegen, also welcher Endpunkt mit welcher Funktion aufgerufen werden soll. In unserer Software haben wir zwei Controller registriert, *PandemieIncAPIController* und *PandemieIncFrontendController*. Der API-Controller verbindet die entworfenen Implementationen mit Spring und der Frontend-Controller gibt eine kleine Startseite mit den verfügbaren Implementation und zusätzlich eine Runden-Visualisierung (siehe Kapitel 4.2.1).

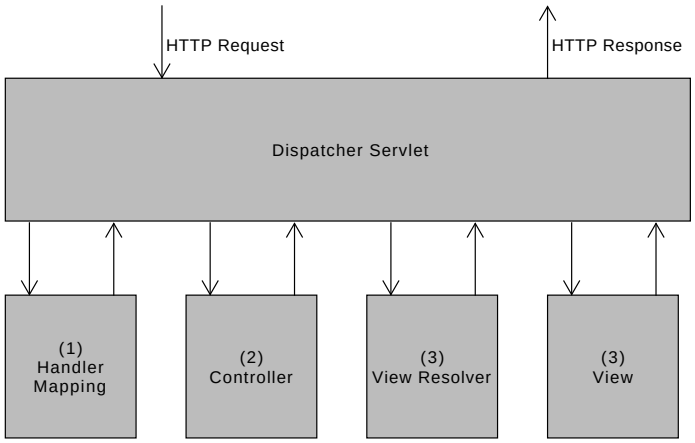


Abbildung 4: Funktionsweise von Spring mit einem HTTP-Request

Die gegebene Runde, welche wir vom Test-Client bekommen, parsen wir in die Klasse *Round*, welche die Städte, Events und Pathogene auch in eigene Klassen parst. Den Aufruf des Parsers übernimmt Spring automatisch, da wir es als Request-Parameter definiert haben im Controller.

Nach diesem Schritt ruft Spring über den Controller eine der definierten Funktionen auf, welche unsere Implementationsklasse erstellt und die Funktion *selectAction* aufruft. Diese

[1] https://spring.io/projects/spring-framework

weählt eine Aktion aus und gibt sie als JSON-String zurück. Dieser String wird von Spring dann an den Test-Client zurückgegeben, womit der Aufruf dann beendet ist und das Spiel entweder abgeschlossen ist oder noch weitergeht.

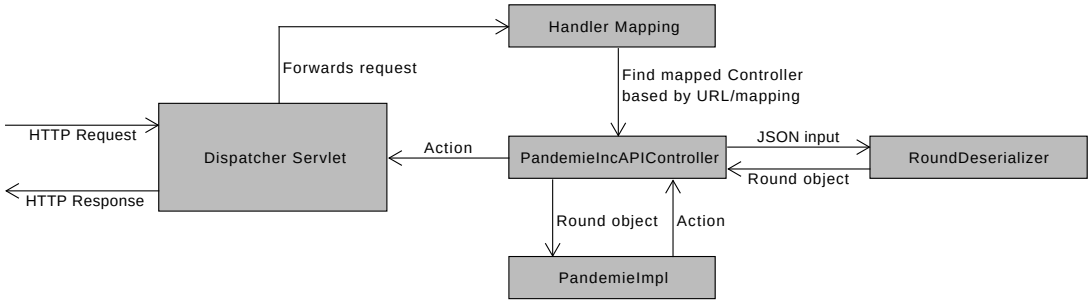


Abbildung 5: Ablauf einer Runde von unserer Software

Außerdem haben wir für das Parsen der Events eine EventFactory erstellt, welche einen JSON-Node von Jackson[2] (JSON-Parser) entgegennimmt und diesen dann in eine Eventklasse parst. Daher sieht man in Abbildung 8 einige Zwischenklassen, welche Events zusammenfassen und dadurch sich der Aufwand beim parsen reduziert.

[2] https://github.com/FasterXML/jackson

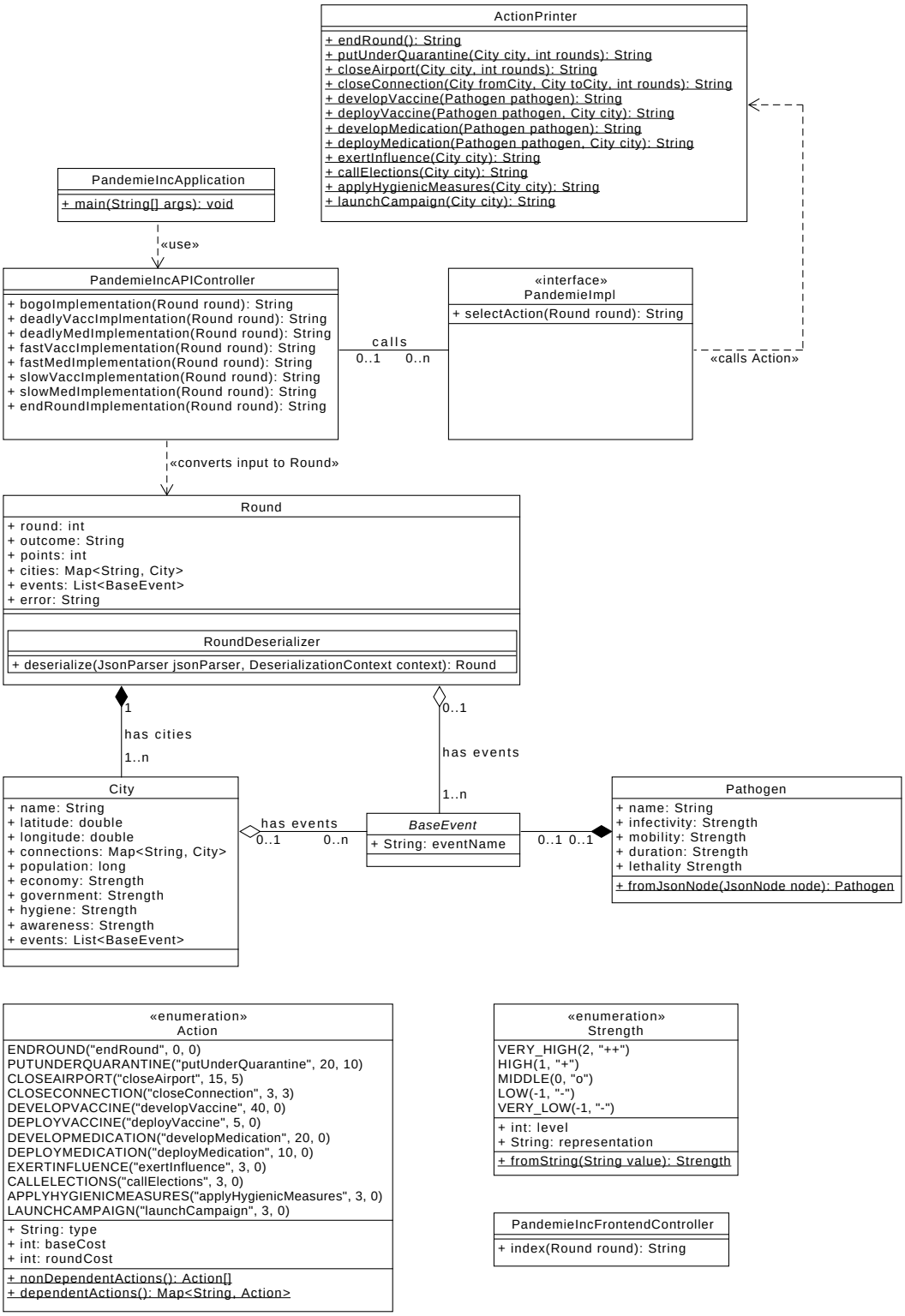


Abbildung 6: UML Klassendiagramm mit den wichtigen Klassen als Übersicht

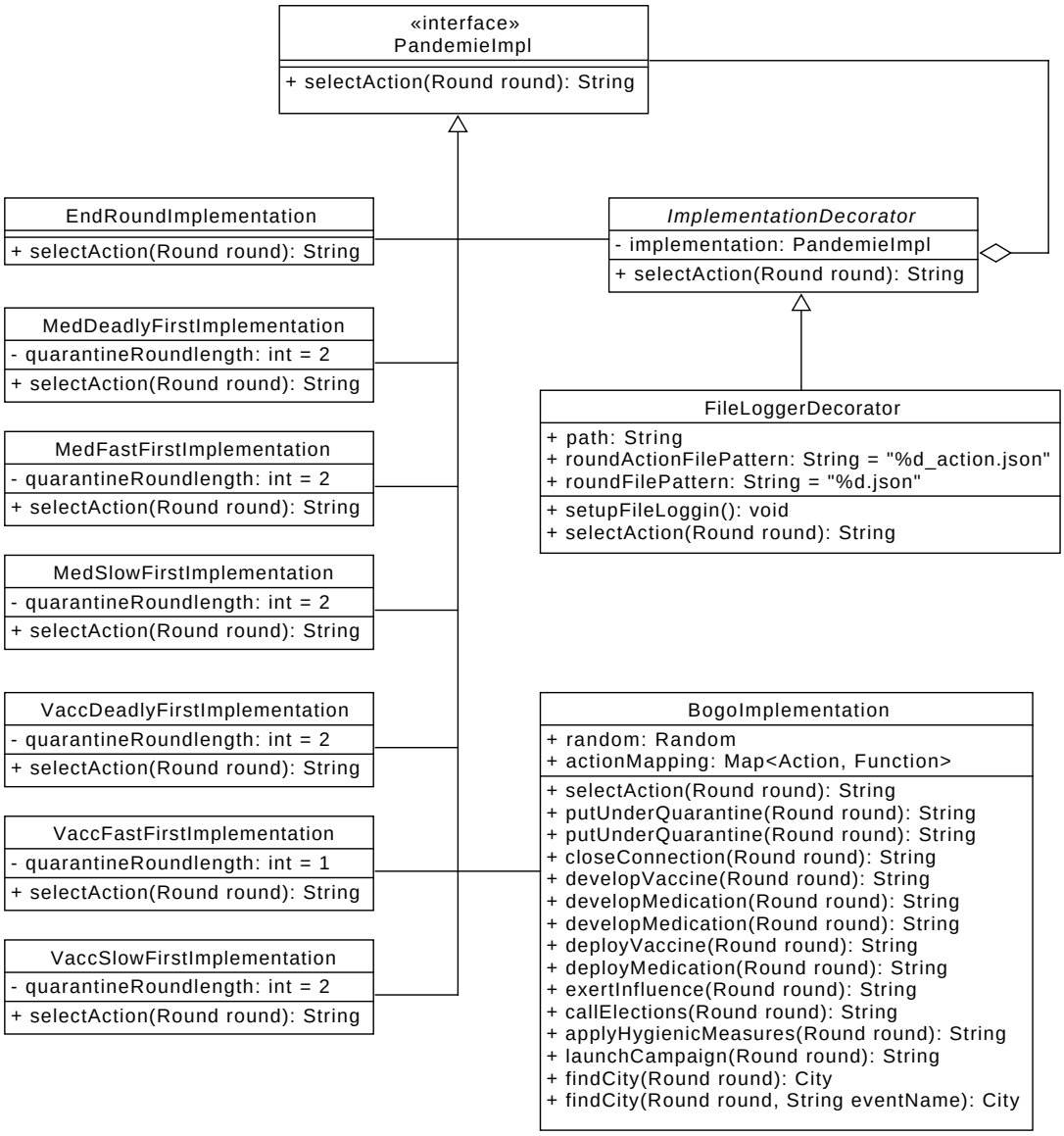


Abbildung 7: UML Klassendiagramm über die Implementationen

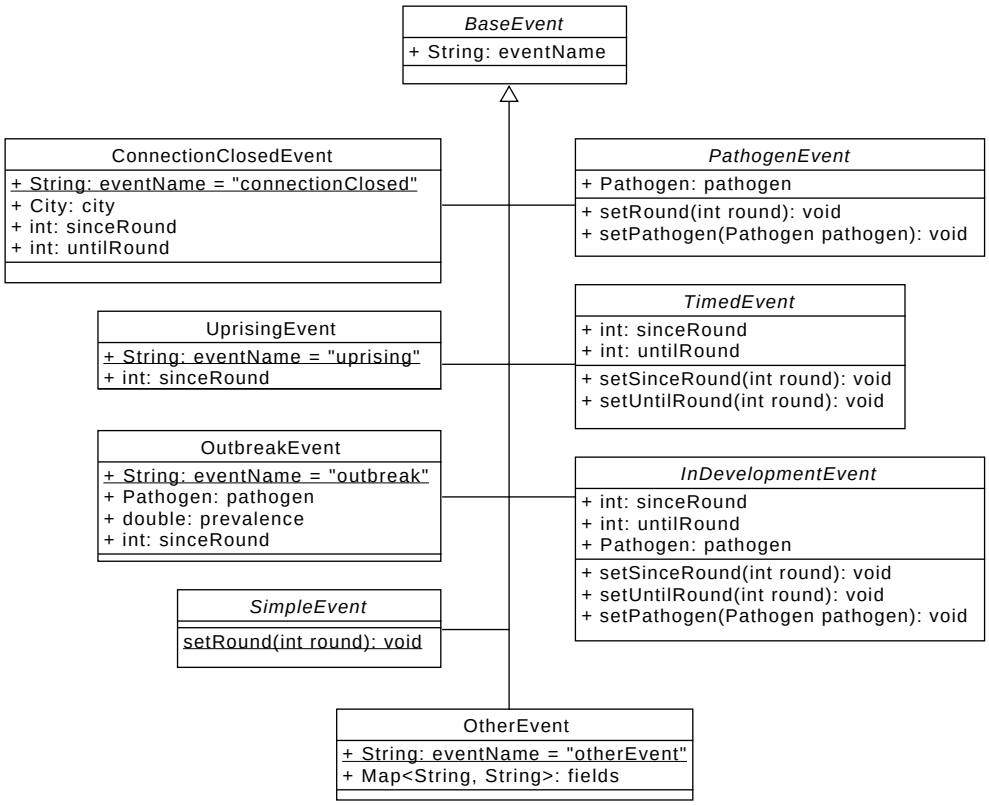


Abbildung 8: UML Klassendiagramm über die Eventklassen (reduzierte Sicht)

3.3 Softwaretesting

Für unsere Klassen *ActionHelper*, *ActionPrinter* und *EventFactory* haben wir Unit Tests geschrieben, welche die gegebenen Komponenten isoliert getestet haben.

Aufgrund von Zeitmangel konnten wir keine Integrationstests schreiben, welche z.B. das Parsen von Runde und Events getestet hätten, sowie das Testen des *FileLoggerDecorator* mit der Implementation. Die Software wurde aber mit dem Testclient auf unterschiedlichen Computern getestet, so dass jede geschriebene Implementation funktioniert und eine entsprechende Aktion bei gegebener Runde zurückliefert.

3.4 Coding conventions

Als Coding conventions haben wir Googles Java Style Guide[3] mit einer Ausnahme festgelegt. Statt mit zwei Leerzeichen einzurücken, verwenden wir 4 Leerzeichen, um die Sichtbarkeit eingerückter Blöcke zu verbessern.

Google stellt freundlicherweise ein Tool bereit (google-java-format[4]), welches den gegebenen Quellcode überprüft und gleichzeitig auch verbessert, bzw. Verbesserungsvorschläge meldet.

Zur Dokumentation des Quellcodes der Implementation verwenden wir Javadoc[5], welches standardmässig in Java integriert ist. Die Dokumentation kann entweder durch den entsprechenden Befehl `javadoc` generiert werden oder durch `mvn javadoc:javadoc` im Projektverzeichnis, wodurch Maven dann javadoc auf die Quellcode-Dateien anwendet. Die generierte Dokumentation kann dann mittels Webbrowser im Verzeichnis `target/site/apidocs/` betrachtet werden.

3.5 Wartbarkeit

Unsere Software ist bezüglich der Wartbarkeit bzw. Anpassbarkeit gut aufgestellt. Wir verwenden vom Spring-Framework nur essentielle Komponenten, da wir keinen Zugriff auf eine Datenbank oder sonstige Quellen benötigen.

Außerdem haben wir mit der Struktur der Eventklassen, bzw. der EventFactory, sichergestellt, dass auch Events auftreten können, welche wir noch nicht entdeckt haben. Diese werden dann in der Klasse *OtherEvent* (siehe Abbildung 8) geparst, welche die Attribute (Key und Value) in einer Map speichert und somit die Implementation trotzdem eine Aktion auswählen kann. Dasselbe gilt auch für die anderen Basisklassen, da man im Übersichtsdiagramm (siehe Abbildung 6) recht gut erkennen kann, wo man ansetzen sollte, falls sich Änderungen in der Kommunikation zwischen Server und Client ergeben.

[3] https://google.github.io/styleguide/javaguide.html
[4] https://github.com/google/google-java-format
[5] https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Implementationen können auch dynamisch verändert werden und das Verhalten der Implementationen kann man dank des Decorator-Patterns (siehe Abbildung 7) verändern, ohne die Implementation im Kern zu verändern, was für kleine Experimente am Algorithmus sehr hilfreich sein kann.

4 Benutzerhandbuch

4.1 Installation

Dieses Kapitel beschreibt alle wesentlichen Inhalte, um unsere Implementation zu kompilieren und zu benutzen.

4.1.1 Abhängigkeiten

Für die erfolgreiche Kompilierung der Software wird folgendes benötigt:

- Java Development Kit[6] oder OpenJDK[7], beide mindestens Version 8
- Apache Maven[8]

Falls es Schwierigkeiten mit der Installation von Maven gibt, liefern wir auch einen Maven Wrapper mit, der alternativ genutzt werden kann. Statt `mvn` nutzt man im Quellcode verzeichnis: `mvnw` , bzw. `mvnw.cmd` unter Windows.

Maven dient als Build-Management-Tool für unser Projekt, welches die benötigten Abhängigkeiten herunterlädt und die Software dann kompiliert und somit ausführbar macht.

4.1.2 Kompilieren der Software

Mittels Maven

Man kann entweder `mvn spring-boot:run` benutzen, welches dann den Quellcode kompiliert und sofort startet, nachdem alle benötigten Abhängigkeiten bereitliegen.

Alternativ kann man durch `mvn package` die Software kompilieren und zu einer Jar-Datei zusammenfassen, die dann mittels Java gestartet werden kann. Diese befindet sich im Ordner `target` (Bsp: `target/pandemieinc-1.0.0-release.jar`).

Mittels Docker

Wir liefern ein Dockerfile mit, welches genutzt werden kann, um ein lauffähiges Docker-Image zu bauen. Dazu muss im Terminal `docker build -t tubs/pandemieinc .` eingegeben werden.

4.1.3 Starten der Software

Mittels Maven

[6] https://jdk.java.net/
[7] https://openjdk.java.net/
[8] https://maven.apache.org/

Falls man keine Jar-Datei generiert hat, kann man die Software mittels `mvn spring-boot:run` starten und direkt nutzen.

Alternativ, falls man eine Jar-Datei generiert hat, kann man diese mittels `java -jar target/pandemieinc-1.0.0-release.jar` starten.

Mittels Docker

Nachdem das Docker-Image gebaut wurde, kann es mittels `docker run -p 8080:8080 tubs/pandemieinc` gestartet werden.

Wenn man die Software erfolgreich gestartet hat, sollte man im Terminal ein Spring-Logo sehen.

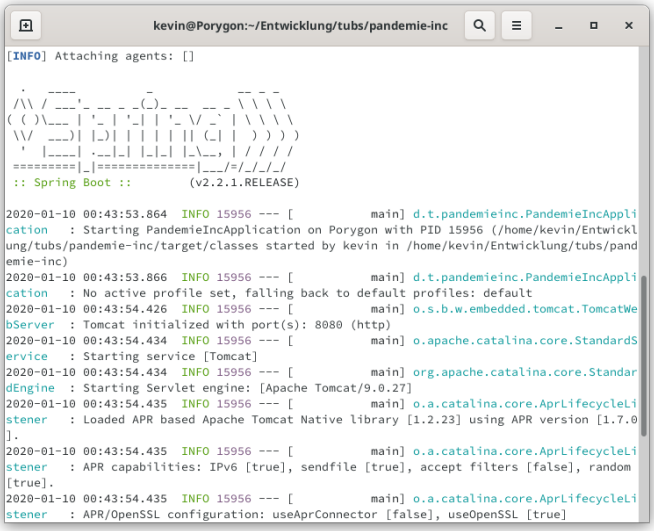


Abbildung 9: Bildschirmfoto: Start der Implementation

4.2 Benutzung der Software

Für die Bewertung geben wir folgende Implementation ab:
MedDeadlyFirstImplementation (`http://localhost:8080/api/deadlyMed`).
Die anderen Implementationen können der Tabelle 4.2 entnommen werden.

Die Implementationen von MedDeadlyFirstImplementation bis VaccSlowFirstImplementation werden im Lösungsansatz 2.2 von der theoretischen Lösung erklärt.

BogoImplementation wählt von den verfügbaren Aktionen, welche sich durch die verfügbaren Punkte in der Runde ergeben, eine Aktion zufällig aus. Diese Implementation wurde zu Dokumentationszwecken benutzt und kann auch benutzt werden um herauszufinden, ob eine Lösungsstrategie für ein gewisses Spiel existiert.

EndRoundImplementation wählt immer die Aktion endRound aus. Dies haben wir zu Dokumentationszwecken benutzt.

TriforceKIImplementation und TriforceKIImplementation2 sind Implementationen, welche mit einem neuronalen Netzwerk auswählen, welche Implementation von MedDeadlyFirstImplementation, MedFastFirstImplementation oder MedSlowFirstImplementation gewählt wird. Die Wahl ist abhängig von den Pathogenen, welche am Start des Spiels erscheinen. Diese Implementation wurden im Zeitraum der Verlängerung der Abgabe entwickelt, wodurch die neuronalen Netzwerke leider kein wirksames Training erhalten konnten.

Tabelle 10: Auflistung der verschiedenen Implementationen

Implementation	**URL für den Endpunkt**
BogoImplementation	`http://localhost:8080/api/bogo`
EndRoundImplementation	`http://localhost:8080/api/endRound`
MedDeadlyFirstImplementation	`http://localhost:8080/api/deadlyMed`
MedFastFirstImplementation	`http://localhost:8080/api/fastMed`
MedSlowFirstImplementation	`http://localhost:8080/api/slowMed`
VaccDeadlyFirstImplementation	`http://localhost:8080/api/deadlyVacc`
VaccFastFirstImplementation	`http://localhost:8080/api/fastVacc`
VaccSlowFirstImplementation	`http://localhost:8080/api/slowVacc`
TriforceKIImplementation	`http://localhost:8080/api/triforce`
TriforceKIImplementation2	`http://localhost:8080/api/triforce2`

4.2.1 Zusatz: Visualisierung eines Spiels

Upload eines Spiels:

Ein Spiel kann per drag-and-drop oder über den Button *select game* ausgewählt werden. Dabei müssen entweder alle Runden in einer einzelnen JSON-Datei gespeichert sein oder jede Runde in einer eigenen Datei. Desweiteren kann eine `seed.txt`, die ausschließlich den Seed enthält, mit übergeben werden (optional). Die JSON-Dateinamen dürfen den String `_action` nicht enthalten. Die Reihenfolge der Dateien und Daten ist nicht relevant.

Aufbau der JSON-Datei, wenn in einer Datei alle Runden gespeichert sind:

```
[
    {
        "outcome":"pending",
        "round":1,
        "points":40,
        "cities":{...},
        "events":[{...}],
    },
    {...},
]
```

Aufbau der JSON-Dateien, wenn je Datei eine Runde gespeichert ist:

```
{
    "outcome":"pending",
    "round":1,
    "points":40,
    "cities":{...},
    "events":[{...}],
}
```

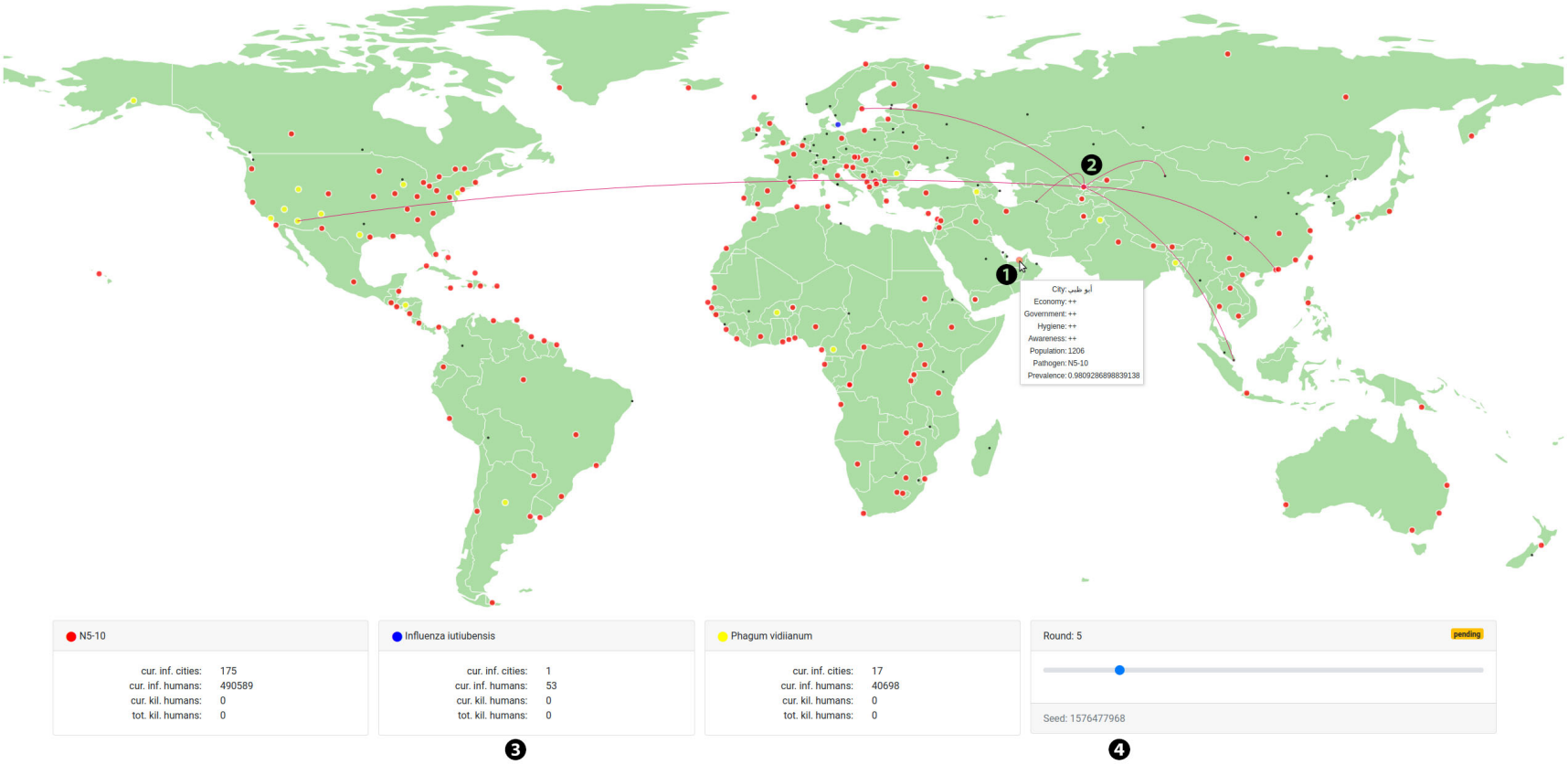


Abbildung 10: Bildschirmfoto vom Frontend

Features:

Jedem Pathogen wird eine einzigartige Farbe zugewiesen. Städte, die mit einem Pathogen infiziert sind, haben große Punkte in derselben Farbe. Städte, die mit keinem Pathogen infiziert sind, haben kleine schwarze Punkte.

1. Wenn die Maus über den Punkt einer Stadt bewegt wird (hover), werden die Stadteigenschaften und ggf. das Pathogen, mit dem diese Stadt infiziert ist, angezeigt.
2. Wenn auf den Punkt einer Stadt geklickt wird, werden alle Verbindungen zu anderen Städten angezeigt. Diese werden ausgeblendet, sobald man auf dieselbe Stadt klickt oder ersetzt, falls man auf eine andere Stadt klickt.
3. Über jedes Pathogen, welches im aktuellen Spiel vorkommt, werden verschiedene Informationen angezeigt:

Wert	**Bedeutung**
cur. inf. cities	Anzahl der aktuell infizierten Städte
cur. inf. humans	Anzahl der aktuell infizierten Menschen ($*10^3$)
cur. kil. humans	Anzahl der in dieser Runde getöteten Menschen
tot. kil. humans	Anzahl der bis zu dieser Runde getöteten Menschen

4. Die aktuelle Runde, das aktuelle Spielergebnis und der Seed (falls verfügbar) werden angezeigt. Die darzustellende Runde kann mit dem Slider oder der rechten und linken Pfeiltaste angepasst werden.

5 Diskussion

Im Folgenden werden ein paar Auswertungen zu den Algorithmen besprochen und auf die Unterschiede aufmerksam gemacht. Wir haben die Implementationen auf 8192 unterschiedlichen Spielen getestet und notiert, ob diese gewonnen oder verloreren haben und nach wie vielen Runden das Spiel beendet wurde.

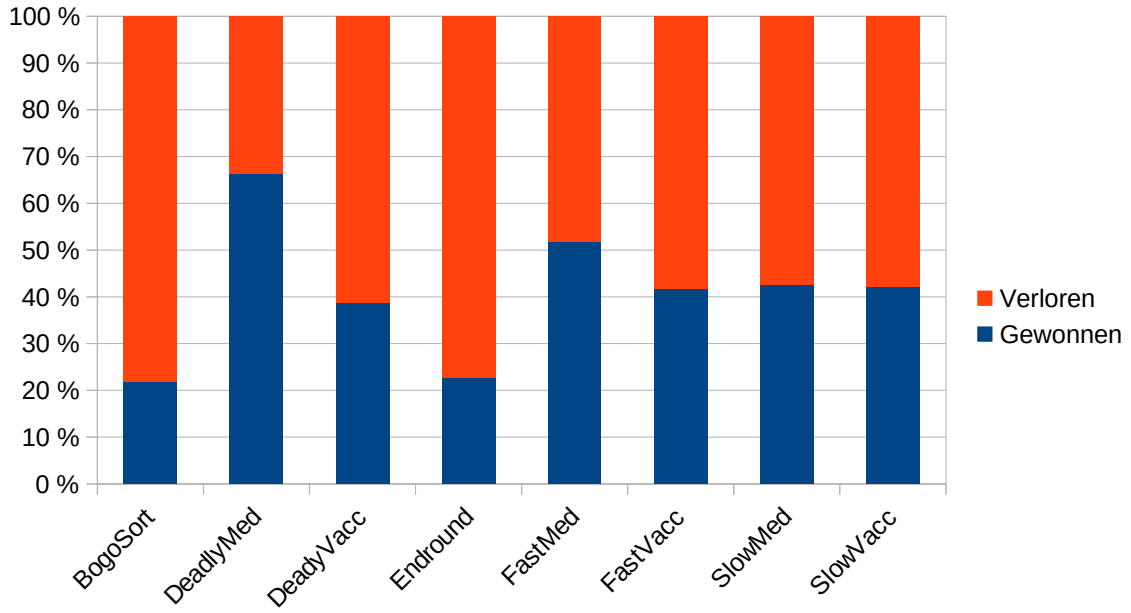


Abbildung 11: Strategien im Vergleich nach Anzahl der Siege

Anhand des obigen Diagramms lässt sich beobachten, dass BogoSort sogar schlechter abschneidet als EndRound. Dies kann daran liegen, dass es nicht immer sinnvoll ist irgendetwas zu machen. Beipielsweise kann es schädlich für den Spielerfolg sein, Quarantäne auf einen schwachen Virus zu setzen, da sich die starken dann besser verbreiten können. Außerdem lässt sich beobachten, dass alle *Med-Strategien besser wirken als die *Vacc-Strategien. Das kann daran liegen, dass es schon zu spät ist für eine Impfung bzw. die Bevölkerung schon immun ist, wenn der Impfstoff entwickelt wurde. Man kann deutlich erkennen, dass DeadlyMed die erfolgreichste Strategie ist.

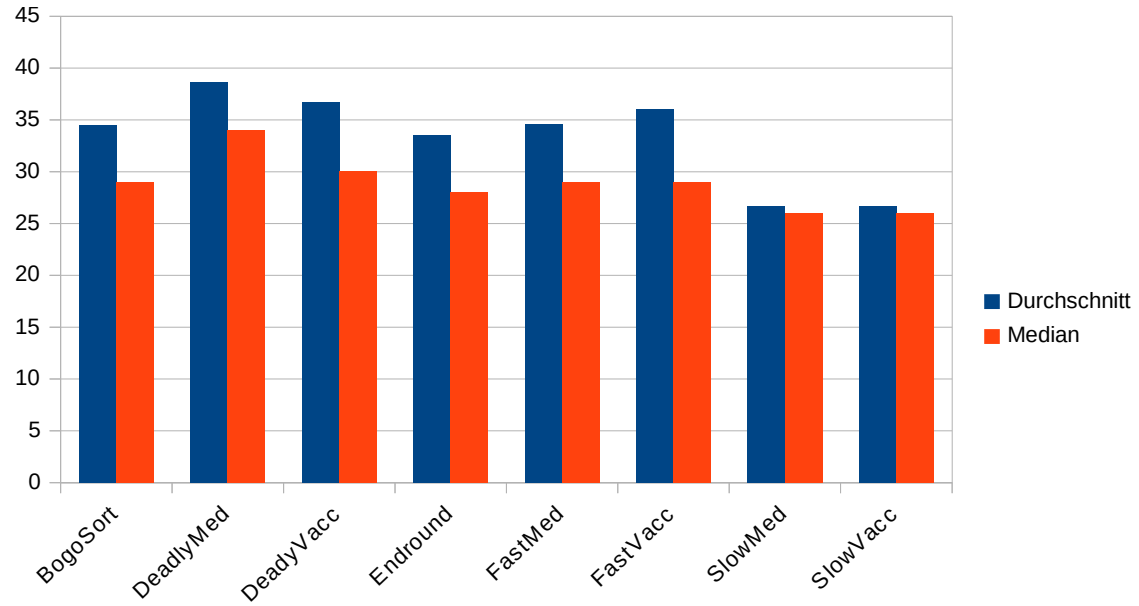


Abbildung 12: Strategien im Vergleich nach durschnittlicher Spielzeit

Die durchschnittliche Spielzeit ist bei DeadlyMed am höchsten. Das könnte daran liegen, dass das Spiel am längsten am Laufen bleibt. Es wurde sich aber dennoch für DeadlyMed entschieden, da die Siegesquote als wichtiger erachtet wurde als die Schnelligkeit. In der Statistik sind nämlich auch alle verlorenen Spiele enthalten und somit kann schlecht auf gute Spiele / Algorithmen anhand dieser zurückschließen.

6 Fazit

6.1 Ausblick

Einige Ideen konnten wir leider aus zeitlichen Gründen nicht umsetzen, dennoch möchten wir diese hier kurz erwähnen.

Das Frontend könnte direkt als Spiel dienen, sodass über WebSockets Frontend und Backend miteinander kommunizieren und der Spieler die Implementation darstellt.

Der Algorithmus, der entscheidet, welche unserer Implementationen ausgewählt wird, hätte noch weiter verbesser werden können. Mit einem optimalen Entscheidungsalgorithmus könnten wir ca. 82% der Spiele gewinnen.

Außerdem wäre Machine Learning als Implementation eine Möglichkeit gewesen. Aufgrund der vielen Variablen hätten wir allerdings viel Zeit und entsprechende Ressourcen oder eine geeignete Trainigsstrategie benötigt.

6.2 Schlusswort

Das Projekt war insgesamt sehr interessant und eine willkommene Abwechslung zum sonst eher theoretischen Uni-Alltag. Es war viel Arbeit, aber wir hatten auch viel Spaß beim Bearbeiten der Aufgabe. Außerdem wurden viele verschiedene Bereiche abgedeckt, in denen wir einiges gelernt haben.

7 Anhang

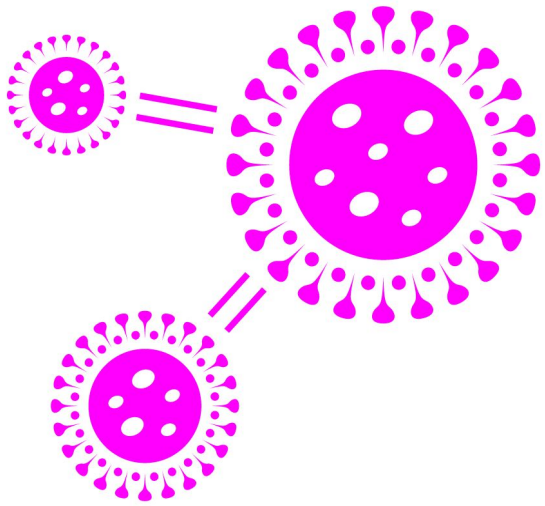
Pandemie!

Keime sind Mikroorganismen oder subzelluläre Erreger, die in anderen Organismen pathogene (d.h. gesundheitsschädigende) Abläufe verursachen. Solche Krankheitserreger können zum Beispiel Bakterien oder Viren sein. Die Ansteckung mit einem Krankheitserreger nennt man Infektion. Voraussetzung für eine Infektion ist die Ausbreitung der Keime, so dass diese mit einem Organismus in Kontakt kommen. Im Organismus erfolgt dann eine Vermehrung als Grundlage für eine weitere Ausbreitung.

Für den Schutz gegen Keime sowie für deren Bekämpfung existieren eine Reihe von möglichen Maßnahmen. Bei einer Quarantäne werden infizierte Organismen isoliert, um die weitere Ausbreitung von Keimen zu verhindern. Impfungen aktivieren das Immunsystem gegen spezifische Keime und beugen so Infektionen vor. Arzneimittel wie zum Beispiel Virostatika oder Antibiotika hemmen die Vermehrung von Keimen oder töten diese sogar ab.

Viele gefährliche Krankheiten wie Cholera, Ebola oder Masern erfordern die effiziente Auswahl und effektive Anwendung möglicher Gegenmaßnahmen. Dieser anspruchsvollen Aufgabe widmet sich der diesjährige InformatiCup.

Aufgabe



Implementieren Sie eine Software, die in einem rundenbasierten Spiel die Menschheit vor der Auslöschung durch eine Pandemie[1] rettet.

Gegeben ist eine endliche Menge von Städten. Eigenschaften der Städte mit unveränderlichen Werten sind Name, Koordinaten und Flugverbindungen zu anderen Städten. Eigenschaften mit veränderlichen Werten sind Einwohnerzahl, Stärke der Wirtschaft, Stabilität der Regierung, Hygienestandards und Achtsamkeit der Einwohner.

[1] https://de.wikipedia.org/wiki/Pandemie

Zu Beginn des Spiels brechen in zufällig gewählten Städten zufällig gewählte Keime aus. Keime haben folgende Eigenschaften, deren Werte alle unveränderlich sind.

- **Name:** Fiktive Bezeichnung des Keims
- **Infektiosität:** Wie hoch ist die Wahrscheinlichkeit, dass ein Infizierter andere Einwohner seiner Stadt oder Einwohner verbundener Städte infiziert?
- **Mobilität:** Wie wahrscheinlich ist der Übergang des Keims auf nicht verbundene Städte?
- **Dauer:** Wie viele Runden dauert die Infektion? Ein Infizierter ist während der gesamten Dauer ansteckend.
- **Letalität**: Wie hoch ist die Wahrscheinlichkeit, dass ein Infizierter stirbt? Infizierte sterben entweder oder werden immun und können nicht mehr mit dem Keim infiziert werden.

Die Menge der Keime ist endlich und Ihnen zunächst nicht bekannt. Nach hinreichend vielen Spielen werden Sie alle Keime gesehen haben. Eigenschaften der Städte und Keime wirken sich auf die Ausbreitung der Keime innerhalb der Städte und zwischen den Städten sowie auf die Eintrittswahrscheinlichkeiten von Ereignissen aus.

Ausbreitung und Auswirkungen der Keime basieren auf einem fiktiven, stark vereinfachtem, stochastischen Modell. Bitte verwenden Sie kein Biologie-Lehrbuch für die Lösung dieser Aufgabe sondern ausschließlich die Ergebnisse Ihrer Beobachtungen innerhalb der Spiele.

In jeder Runde können Sie beliebig viele Aktionen durchführen (siehe Abschnitt *Ausgabe*). Jede Aktion kostet Punkte. Sie beginnen ein Spiel mit 40 Punkten Startguthaben und erhalten pro Runde 20 weitere Punkte. Das Spiel endet, wenn alle Keime eliminiert wurden oder die Menschheit ausgelöscht wurde. Die Menschheit gilt als ausgelöscht, wenn am Ende einer beliebigen Runde die Summe der Einwohner aller Städte gegenüber der ersten Runde (mehr als) halbiert wurde.

Eingabe

Ihre Software muss einen Webservice bereitstellen, über den die Eingabe der Spielzustände per HTTP-POST-Requests erfolgt. Im Body der Requests sind Objekte in JSON[2] (UTF-8) wie in dem folgenden Beispiel enthalten.

Die numerischen Eigenschaftswerte der Städte und Keime werden auf die Angaben "--" (sehr niedrig), "-", "o", "+" und "++" (sehr hoch) abgebildet.

[2] https://de.wikipedia.org/wiki/JavaScript_Object_Notation

```
{
  "round": 4, // Aktuelle Runde, zu Beginn des Spiels 1
  "outcome": "pending", // Ausgang des Spiels, "win" | "loss" | "pending"
  "points": 20, // Verfügbare Punkte, natürliche ganze Zahl
  "cities": { // Mindestens 2 Einträge
    "Hamburg": { // Name der Stadt, eindeutig über alle Städte
      "latitude": 53.548450, // Längengrad
      "longitude": 9.978514, // Breitengrad
      "population": 1822, // 10^3 Einwohner, natürliche ganze Zahl
      "connections": ["Berlin", "Köln", "München", ...], // Flugverbindungen
      "economy": "++", // Stärke der Wirtschaft
      "government": "+", // Stabilität der Regierung
      "hygiene": "++", // Hygienestandards
      "awareness": "+", // Achtsamkeit der Einwohner
      "events": [ // Ereignisse (siehe den Hinweis zu Ereignistypen)
        {
          "type": "outbreak", // Ausbruch eines Keims
          "sinceRound": "4", // Runde, in der das Ereignis eingetreten ist
          "pathogen": {
             "name": "Coccus innocuus", // Name des Keims
             "infectivity": "+", // Infektiosität
             "mobility": "-" // Mobilität
             "duration": "-", // Dauer
             "lethality": "--", // Letalität
          },
          "prevalence": 0.34, // Anteil der infizierten Einwohner der Stadt
        },
        {
          "type": "uprising", // Aufstand!
          "sinceRound": "4", // Runde, in der das Ereignis eingetreten ist
          "participants": 36, // 10^3 Teilnehmer, natürliche ganze Zahl
        }
      ]
    },
    "Berlin": { ... }, ...
  ],
  // Siehe Felder "events" der Städte. Diese Ereignisse betreffen das gesamte
  // Spiel und nicht nur einzelne Städte.
  "events": [ ... ],
  // Feld "error" ist nur enthalten, wenn eine Aktion nicht akzeptiert wurde.
  "error": "..." // Natürlichsprachliche Beschreibung
}
```

Spielzustände können zu unterschiedlichen Spielen gehören. Da sie nicht eindeutig einem Spiel zugeordnet werden können, muss Ihre Software in der Lage sein, Entscheidungen ohne Rückgriff auf vorige Spielzustände zu treffen.

Die Menge der Ereignistypen ist (analog zur Menge der Keime) endlich und Ihnen zunächst unbekannt. Dies umfasst die unterschiedlichen Eigenschaften der Ereignistypen und deren Auswirkungen auf das Spielgeschehen. Manche Ereignisse lassen Rückschlüsse auf vorige Spielzustände zu, beispielsweise seit welcher Runde ein Impfstoff gegen einen Keim entwickelt wird.

Berücksichtigen Sie die Umsetzung der Produktivversion Ihrer Software als zustandslosen Webservice von Anfang an in Ihrem Softwareentwurf.

Ausgabe

Auf eine Eingabe des Spielstandes muss Ihr Webservice mit einem Aktions-Objekt in JSON (UTF-8) wie in dem folgenden Beispiel antworten.

```
{
  "type": "closeAirport", // Aktionstyp laut Tabelle
  "city": "Berlin", // Sonstige Felder laut Tabelle
  "rounds": 4
}
```

Im Folgenden finden Sie eine vollständige Aufstellung der Ihnen zur Verfügung stehenden Aktionen.

Name	Format	Kosten (Punkte)
Runde beenden	`{"type": "endRound"}`	0
Die aktuelle Runde wird beendet. Es können in dieser Runde keine weiteren Aktionen ausgeführt werden. Der nächste Spielzustand wird eingegeben.		
Quarantäne anordnen	`{"type": "putUnderQuarantine", "city": "<Name einer Stadt: S>", "rounds": "<Anzahl Runden: R, natürliche Zahl > 0>"}`	10 × Anzahl Runden + 20
Keime breiten sich in den nächsten *R* Runden nicht von *S* in andere Städte aus.		
Flughafen schließen	`{"type": "closeAirport", "city": "<Name einer Stadt: S>", "rounds": "<Anzahl Runden: R, natürliche Zahl > 0>"}`	5 × Anzahl Runden + 15
Keime breiten sich in den nächsten *R* Runden nicht über Flugverbindungen von und nach *S* aus.		
Flugverbindung sperren	`{"type": "closeConnection", "fromCity": "<Name einer Stadt: S1>", "toCity": "<Name einer Stadt: S2>", "rounds": "<Anzahl Runden: R, natürliche Zahl > 0>"}`	3 × Anzahl Runden + 3
Keime breiten sich in den nächsten *R* Runden nicht über die Flugverbindung von *S1* nach *S2* aus.		
Impfstoff entwickeln	`{"type": "developVaccine", "pathogen": "<Name eines Keims: K>"}`	40
In der 6. Runde nach Ausführung dieser Aktion steht ein Impfstoff gegen den Keim *K* zur Verfügung. Voraussetzung: *K* muss im Laufe des Spiels bereits ausgebrochen sein.		
Impfstoff verteilen	`{"type": "deployVaccine", "pathogen": "<Name eines Keims: K>", "city": "<Name einer Stadt: S>"}`	5
Alle nicht infizierten Einwohner der Stadt *S* werden sofort immun gegen den Keim *K*. Voraussetzung: Ein Impfstoff gegen den Keim *K* steht zur Verfügung.		

Medikament entwickeln	`{"type": "developMedication", "pathogen": "<Name eines Keims: K>"}`	20
In der 3. Runde nach Ausführung dieser Aktion steht ein Medikament gegen den Keim *K* zur Verfügung. Voraussetzung: Der Keim *K* muss im Laufe des Spiels ausgebrochen sein.		
Medikament verteilen	`{"type": "deployMedication", "pathogen": "<Name eines Keims: K>", "city": "<Name einer Stadt: S>"}`	10
30% bis 50% der mit dem Keim *K* infizierten Einwohner der Stadt *S* werden geheilt und sind in der Folge immun gegen *K*. Voraussetzung: Ein Medikament gegen den Keim *K* steht zur Verfügung.		
Politischen Einfluss geltend machen	`{"type": "exertInfluence", "city": "<Name einer Stadt: S>"}`	3
Der Eigenschaftswert *Stärke der Wirtschaft* der Stadt *S* wird zufällig neu gesetzt.		
Neuwahlen ausrufen	`{"type": "callElections", "city": "<Name einer Stadt: S>"}`	3
Der Eigenschaftswert *Stabilität der Regierung* der Stadt *S* wird zufällig neu gesetzt.		
Hygienemaßnahmen durchführen	`{"type": "applyHygienicMeasures", "city": "<Name einer Stadt: S>"}`	3
Eigenschaftswert *Hygienestandards* der Stadt *S* wird zufällig erhöht.		
Informations-kampagne starten	`{"type": "launchCampaign", "city": "<Name einer Stadt: S>"}`	3
Der Eigenschaftswert *Achtsamkeit der Einwohner* der Stadt *S* wird zufällig erhöht.		

Testen

Im GitHub-Repository des aktuellen Wettbewerbs[3] finden Sie ein Kommandozeilenwerkzeug[4] als ausführbare Datei für Windows, macOS und Linux (x86-64). Mit Hilfe dieses Werkzeugs können Sie Ihre Implementierung testen. Alle Funktionen des Kommandozeilenwerkzeugs können Sie mit dem Parameter `--help` erfragen. Im GitHub-Repository finden Sie auch eine Beispiellösung, die Ihnen als Orientierungshilfe dienen kann. Die Beispiellösung stellt einen Webservice bereit, der zu jeder Eingabe des Spielstandes die Aktion *Runde beenden* ausgibt.

[3] https://github.com/InformatiCup/InformatiCup2020
[4] https://github.com/InformatiCup/InformatiCup2020/releases/latest

Deployment

Die Juroren werden Ihre Software als Webservice verwenden. Dazu müssen Sie Ihre Software bitte entsprechend wie folgt unmittelbar erstellbar und ausführbar bereitstellen.

1. Sie verwenden ein beliebiges Build-System, das Erstellung und Ausführung Ihrer Software mit einem Befehl ermöglicht, oder (XOR)
2. sie stellen ein Dockerfile zur Verfügung oder (XOR)
3. sie stellen einen ausführbaren Webservice auf einer Plattform Ihrer Wahl zur Verfügung.

Eine Beschreibung wie Sie mit AWS Lambda[5] deployen können finden Sie hier: https://docs.aws.amazon.com/lambda/latest/dg/deploying-lambda-apps.html

Das Deployment Ihrer Software ist in diesem Jahr ein Teil der Aufgabenstellung. Die entsprechende Verfügbarkeit Ihrer Lösung ist Voraussetzung für deren Bewertung.

Die Juroren werden Ihre Software mit dem unter *Testen* beschriebenen Kommandozeilenwerkzeug testen.

Bewertung

Ihre Software sollte innerhalb möglichst weniger Runden alle Keime eliminieren. Lösung **A** wird höher bewertet als Lösung **B**, wenn…

- **A** alle Keime eliminiert und bei **B** die Menschheit auslöscht wird.
- **A** und **B** beide alle Keime eliminieren und **A** weniger Runden benötigt.
- Bei **A** und **B** die Menschheit ausgelöscht wird und bei **A** dafür mehr Runden benötigt werden.

Neben der Software erstellen Sie bitte eine Ausarbeitung, die die Installation und Bedienung Ihrer Software sowie Ihren theoretischen Lösungsansatz beschreibt. Ihre Einreichung wird ganzheitlich bewertet. Neben der Güte der Lösung werden der theoretische Lösungsansatz, die Form der Ausarbeitung, die Softwarearchitektur und -qualität, mögliche Erweiterungen und, wenn Sie Ihre Lösung im Finale vorstellen dürfen, die Qualität der Präsentation bewertet.

Im Anhang finden Sie eine Checkliste der Bewertungskriterien. Nutzen Sie diese Liste, um die Vollständigkeit Ihrer Lösung zu überprüfen.

[5] https://en.wikipedia.org/wiki/AWS_Lambda

Außerdem

Die FAQs zu dieser Aufgabenstellung finden Sie in Kürze online in dem GitHub-Repository https://github.com/InformatiCup/InformatiCup2020.

Checkliste Bewertungskriterien

Theoretischer Ansatz

Der theoretische Ansatz muss in einer Ausarbeitung, die zusammen mit der Implementierung eingereicht wird, dargestellt werden. Bewertet werden sowohl der Inhalt als auch die Form.

Theoretische Ausarbeitung

- ❏ **Hintergrund**: Welche theoretischen Ansätze wurden verwendet? Warum wurden diese Ansätze verwendet?
- ❏ **Auswertung**: Wie hoch ist die Güte der Lösung?
- ❏ **Diskussion**: Wie lässt sich die Güte der Lösung erklären und erhöhen?
- ❏ **Quellen**: Wurden wissenschaftliche Quellen richtig und angemessen verwendet?

Formalien

Eine gute Form ist entscheidend für die Lesbarkeit einer Ausarbeitung. Beachten Sie deshalb neben dem reinen Inhalt Ihrer Ausarbeitung bitte auch einige Formalien.

- ❏ **Rechtschreibung**: Rechtschreibung und Grammatik sind korrekt.
- ❏ **Struktur**: Eine klar erkennbare Struktur wird konsequent verfolgt.
- ❏ **Layout**: Das Dokument hat ein einheitliches Layout. Dieses kann frei gewählt werden, darf aber nicht den Lesefluss stören.
- ❏ **Zitate**: Es wird richtig und einheitlich zitiert.
- ❏ **Quellenangaben**: Quellen sind richtig und einheitlich angegeben.

Softwarearchitektur und -qualität

Da eine etablierte Softwarearchitektur nur mit hohem Aufwand zu ändern ist, sollte sie besonders gründlich durchdacht und ausführlich begründet werden. Ausgewählte Aspekte der Softwarequalität sind für die Bewertung von besonderer Bedeutung. Gerne dürfen auch hier nicht genannte Aspekte aus den sehr weiten Feldern Softwarearchitektur und -qualität beleuchtet werden.

- ❏ **Architektur**: Beschreibung der Komponenten und deren Beziehungen
- ❏ **Software testing**[6]: Begründetes Konzept, Umsetzung
- ❏ **Coding conventions**[7]: Begründetes Konzept, Umsetzung
- ❏ **Wartbarkeit**: Mit welchem Aufwand kann das System angepasst werden?

Handbuch

Das Handbuch beschreibt den Installationsprozess der Lösung als Webservice, insbesondere die Abhängigkeiten. Zudem wird die Benutzung erläutert. Empfohlen wird die genaue Angabe der erforderlichen Befehle oder die Bereitstellung eines Installations-Skripts.

Erweiterungen

Erweitern Sie Ihre Lösung über die Anforderungen der Aufgabenstellung hinaus. Seien Sie kreativ!

Präsentation

Im InformatiCup-Finale werden die besten Lösungen vor einer Fachjury präsentiert.

- ❏ **Foliendesign**: Sind die Folien ansprechend? Lenken sie nicht vom Inhalt der Präsentation ab?
- ❏ **Vortragsstil**: Weckt der Vortragsstil Interesse an der Präsentation?
- ❏ **Verständlichkeit**: Ist der Vortrag verständlich? Wird der Hintergrund der Lösung in einem angemessenen Tempo erklärt? Wird nötiges Vorwissen geschaffen?
- ❏ **Reaktion auf Nachfragen**: Können Nachfragen beantwortet werden?

[6] https://en.wikipedia.org/wiki/Software_testing

[7] https://en.wikipedia.org/wiki/Coding_conventions