BURNINGTrain

Theoretische Ausarbeitung InformatiCup 2020

Pandemie

Autoren
Danial Mirrezaei, Maik Gode,
Mirwais Hotak und Sebastian Volk

19. Januar 2020

Übersicht Leistungserbringung

Gruppenmitglied	Kapitel
Mirwais Hotak	Bedienungsanleitung, 2, 2.1, 2.2, 2.3
Maik Gode	1, 2.4, 3, 6
Sebastian Volk	4, 5
Danial Mirrezaei	2.5

Die Gruppe möchte hiermit ausdrücklich betonen, dass alle Gruppenmitglieder an allen Kapiteln beteiligt gewesen sind und die Softwareanwendung durch gemeinsame Arbeit entstanden ist, bei der jedes Gruppenmitglied maßgeblich bei der Entwicklung beteiligt gewesen ist. Aus der Tabelle geht hervor, welches Gruppenmitglied, welches Kapitel verfasst hat. Aus der Commit History des GitHub Repository geht hervor, welches Gruppenmitglied was programmiert hat. Allerdings ist zu berücksichtigen, dass die Gruppe kontinuierlich Pair-Programming betrieben hat.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Bedienungsanleitung

Repository klonen

Um den Build-Prozess starten zu können, muss das Repository geklont werden.

```
git clone https://github.com/sebastianvolk/informatiCup2020.git

cd informatiCup2020
```

Kompilieren des Quellcodes

Als Build-Tool wird Apache Maven verwendet. Für das Kompilieren ist es daher notwendig, dass Maven und das Java Development Kit installiert sind. Durch den folgenden Befehl kann dann das Programm kompiliert werden.

```
mvn clean package
```

Nach erfolgreichem Kompilieren des Quellcodes befindet sich das ausführbare Programm im „target“ Ordner.

```
cd target
```

Kompiliertes Programm herunterladen

Es besteht auch die Möglichkeit, das ausführbare Programm als jar Datei herunterzuladen. Die aktuelle Version kann unter latest release eingesehen werden.

Ausführen und Server starten

Durch das Ausführen des kompilierten Programms wird der Server gestartet. Voraussetzung dafür ist eine installierte Version der Java Runtime Environment.

```
java -jar pandemicfighter-<version>-jar-with-dependencies.jar
```

Die Variable <version> gibt an, welche Version des Spiels heruntergeladen, bzw. kompiliert wurde.

Das Spiel, dass unter latest release zum Herunterladen zur Verfügung steht, kann ausgeführt werden, nachdem die Kommandozeile die folgende Erfolgsmeldung ausgibt.

```
Server started on port 50123

Quit the server with CONTROL-C
```

1 Einleitung

Dieses Projekt ist im Rahmen des informatiCup 2020 entstanden. Innerhalb des Projekts wird das Ziel verfolgt, die menschliche Spezies vor dem Aussterben als Folge einer Pandemie zu bewahren. Binnen einer virtuellen Welt existieren Keime, die den menschlichen Körper befallen können und Infektionen auslösen.

Die virtuelle Welt wird via bereitgestellter CLI angesprochen. In dieser Welt existieren Städte, Keime und Events. Die Städte verfügen neben ihren persistenten Eigenschaften, wie Name, Koordinaten und Flugverbindungen, über dynamische Eigenschaften, wie zum Beispiel dem Hygienestandard der Bevölkerung oder der Stabilität einer amtierenden Regierung. Die veränderbaren Eigenschaften der Städte beeinflussen die Resistenz bzw. Wahrscheinlichkeit, mit der sich Keime innerhalb der Bevölkerung ausbreiten können und zu Infektionen führen. Krankheitserreger verfügen über vier Eigenschaften, die sich nicht verändern.

Bei dem eigentlichen Simulator handelt es sich um ein Spiel, das rundenbasiert ist. Das Spiel gilt als verloren, sobald 50% Menschen innerhalb des Spiels gestorben sind und die menschliche Rasse folglich als ausgestorben gilt. Das Spiel gilt als gewonnen, sobald alle im Spiel aufgetauchten Keime eliminiert wurden.

Ziel des Projekts ist es, eine Softwareanwendungen zu programmieren, die dazu imstande ist, innerhalb der Runden, vordefinierte Aktionen durchzuführen, die die Spielhandlung beeinflussen und zur Rettung der Welt beitragen, in dem vorgebeugt wird, dass gesunde Menschen von Krankheiten befallen werden oder kranke Menschen von der jeweiligen Krankheit geheilt werden. Im Fokus steht dabei, dass die Anwendung über ein Mindestmaß an „Intelligenz“ verfügen muss, um passende Aktionen in jeweiligen Situationen auszuwählen. Die Intelligenz wird dabei über ein Scoring-Modell bewerkstelligt. Durch das Scoring-Modell kann die Anwendung unterschiedliche Parameter vergleichen, um eine Entscheidung zu treffen. Das Scoring-Modell wird im späteren Ablauf der Dokumentation näher verdeutlicht.

2 Hintergrund

Um eine adäquate Lösung zu entwickeln, wurde die bereitgestellte Welt auf ihre Eigenschaften und Ausprägungen untersucht. Im Vordergrund stand, zu ermitteln, welche Indikatoren sich auf das Ausbrechen von Keimen auswirken. Des Weiteren musste festgestellt werden, wie und auf welcher Grundlage sich Keime nach Ihrer Entdeckung verbreiten bzw. wie sie ausgemerzt werden können. Ferner müssen sämtliche Ausprägungen der Welt analysiert werden, sodass Kennzahlen erstellt werden können, die die entwickelte Software evaluiert, um Entscheidungen zu treffen.

2.1 City

Cities beherbergen die Einwohner, die mit den Keimen infiziert werden. Abgesehen vom Namen, wirken sich sämtliche Eigenschaften einer Stadt darauf aus, wie schnell sich ein Pathogen in der Welt verbreitet.

- *Koordinaten:* Die Koordinaten (Längen- und Breitengrad) kennzeichnen, wo sich eine City, geographisch betrachtet, befindet. Die Wahrscheinlichkeit, dass sich ein in City X ausgebrochenes Pathogen A auf eine City Y ausbreitet ist ohne Berücksichtigung anderer Werte höher, desto geringer die Distanz zwischen den Cities ist.
- *Flugverbindung:* Die Flugverbindungen geben an, welche Cities sich von einer City X direkt erreichen lassen. Den Beobachtungen zufolge ist die Wahrscheinlichkeit, dass sich ein Pathogen A, das in City X ausgebrochen ist, auf City Y ausbreitet höher, wenn von City X und City Y ein Direktflug möglich ist. Dies impliziert, dass weniger bzw. keine Zwischenflüge die Wahrscheinlichkeit der Ausbreitung zwischen zwei Cities verringert.
- *Einwohnerzahl:* Die Einwohnerzahl ist eine dynamische Zahl, die variiert. Je mehr Einwohner eine City beherbergt, desto kritischer ist das Ausbrechen eines Pathogens, da die Krankheit effektiver verbreitet wird.
- *Stärke der Wirtschaft:* Desto schwächer die Wirtschaft einer City X, desto effektiver kann sich ein Pathogen A innerhalb der City verbreiten.
- *Stabilität der Regierung:* Desto instabiler die Regierung einer City X, desto effektiver kann sich ein Pathogen A innerhalb der City verbreiten.
- *Hygienestandards:* Desto höher der Hygienestandard einer City X, desto ineffektiver kann sich ein Pathogen A innerhalb der City verbreiten.
- *Achtsamkeit der Einwohner:* Desto höher die Achtsamkeit der Einwohner einer City X, desto ineffektiver kann sich ein Pathogen A innerhalb der City verbreiten.

2.2 Pathogen

Pathogene sind Keime, die Menschen infizieren. Die Keime verfügen neben ihren Namen über die folgenden persistenten Eigenschaften, die unterschiedlich ausgeprägt seien können:

- *Infektiosität*
- *Mobilität*
- *Dauer*
- *Letalität*

2.3 Events

Innerhalb des Spiels finden diverse Events statt. Um herauszufinden, welche Events existieren, wurde das Spiel einige Male ohne Durchführung von Aktionen ausgeführt. Dabei konnte beobachtet werden, dass unterschiedliche Event-Typen existieren.

City-Events finden in den Städten statt. Dabei betreffen sie zunächst nur die Einwohner der betroffenen Stadt. Folgende City-Events konnten beobachtet werden:

- *BioTerrorism*
- *EconomicCrisis*
- *AntiVaccinationism*
- *LargeScalePanic*
- *Uprising*
- *Outbreak*

Global-Events betreffen nicht nur eine Region bzw. Stadt. Sie finden regionalübergreifend statt und wirken sich auf die gesamte Welt aus. Folgende Global-Events konnten beobachtet werden:

- *MedicationInDevelopment*
- *VaccineInDevelopment*
- *MedicationAvailable*
- *VaccineAvailable*

Neben dem Bereich, auf den sich Events auswirken, lassen sich Events dadurch kategorisieren, ob sie den Spielfluss positiv, negativ oder neutral beeinflussen. Das Event *pathogenEncountered* wirkt sich zum Beispiel neutral auf den Spielfluss aus.

Da die Software über kein Gedächtnis verfügt, das Informationen, wie Events speichert, muss die Software in der Lage sein, beim Auftreten unbekannter Events keine Exception zu werfen, sondern den Benutzer darüber zu informieren, dass ein neues, unbekanntes Event aufgetreten ist.

2.4 Aktionen

Aus der Aufgabenstellung geht hervor, dass in den einzelnen Runden Aktionen durchgeführt werden müssen, wodurch der Spielverlauf verändert wird. Die folgenden Aktionen senken die Geschwindigkeit, mit der sich Keime in der Welt oder einer jeweiligen Region ausbreiten:

- *Quarantäne anordnen:* Durch die Anordnung einer Quarantäne wird vorgebeugt, dass sich eine ausgebrochene Krankheit A über die Grenzen einer Stadt X verbreitet. Diese Aktion ist am effektivsten, wenn die besagte Krankheit A bisher nur in Stadt X vorgefunden wurde. Durch die Durchführung dieser Aktion würde folglich die Wahrscheinlichkeit sinken, dass sich die Krankheit auf eine andere Stadt Y verbreitet.
- *Flughafen schließen:* Durch die Schließung des Flughafens wird verhindert, dass sich Krankheiten über den Flugverkehr verbreiten. Dies reduziert die Geschwindigkeit, mit der sich eine Krankheit verbreitet.
- *Flugverbindung sperren:* Das Sperren von Flugverbindungen führt ebenfalls dazu, dass sich die Geschwindigkeit der Ausbreitung einer Krankheit reduziert. Die Reduzierung ist dabei geringer als bei der Schließung eines Flughafens.
- *Hygienemaßnahmen durchführen:* Durch die Durchführung einer Hygienemaßnahme sinkt die Wahrscheinlichkeit dafür, dass ein Einwohner von einem Keim infiziert wird.
- *Informationskampagne starten:* Durch die Durchführung von Informationskampagnen sinkt die Wahrscheinlichkeit dafür, dass ein Einwohner von einem Keim infiziert wird.

Folgende Aktionen sind dazu imstande Keime auszumerzen bzw. die Verbreitung der Krankheit zu stoppen:

- *Medikament verteilen:* Das Verteilen von Medikamenten führt bei den erkrankten Patienten zu einer sofortigen Heilung und einer anschließenden Immunität gegen die Krankheit. Dabei werden zwischen 30% und 50% der Infizierten geheilt.
- *Impfstoff verteilen:* Das Verteilen von Impfstoffen führt bei den betroffenen Einwohnern zu einer sofortigen Immunität gegen eine Krankheit. Durch das Verteilen von Impfstoffen wird folglich die Verbreitungsgeschwindigkeit einer Krankheit deutlich reduziert.

Darüber hinaus existieren weitere Aktionen, die als Voraussetzung gelten, damit zum Beispiel Medikamente oder Impfstoffe verteilt werden können. Diese wirken sich zunächst nicht aktiv auf den Spielfluss aus.

- *Impfstoff entwickeln:* Gilt als Voraussetzung, damit ein Impfstoff verteilt werden kann, falls noch keines existiert.
- *Medikament entwickeln:* Gilt als Voraussetzung, damit ein Medikament verteilt werden kann, falls noch keines existiert
- *Runde beenden:* Beendet eine Runde.

Die Durchführung von Aktionen ist in dieser Welt die einzige Möglichkeit, den Spielfluss aktiv zu beeinflussen. Der Aktions-Auswahlmechanimus ist daher von essenzieller Bedeutung. Als

Grundlage für die Aktionsauswahl dient ein Bewertungssystem. Auf dessen Grundlage entscheidet die Software, zu welchem Zeitpunkt (Runde), welche Aktion die effiziente Wahl ist. Das Bewertungssystem wird im folgendem Kapitel 2.5 detailliert erläutert.

2.5 Logik

Die Logik der Software muss rational und situationsbedingt darüber entscheiden können, welche Aktion/en zu einem jeweiligen Zeitpunkt die effektivste/n ist/sind. In Abbildung 1 wird der Prozess dargestellt, wie die Software eine Aktion auswählt.

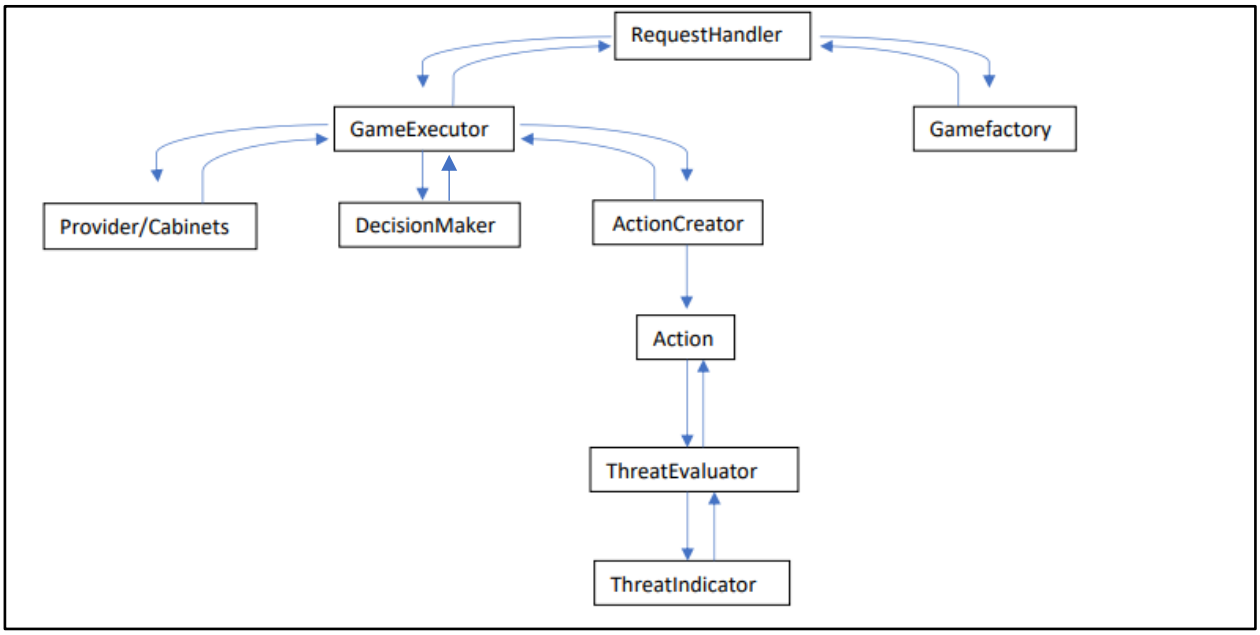

Figure 1: Auswahlprozess einer Aktion

Sobald ein Post-Request im RequestHandler eingeht, wird zunächst durch Zunahme der GameFactory ein Game Objekt aus der überlieferten JSON Datei erzeugt. Anschließend wird ein GameExecutor Obejekt erzeugt und mit dem vorher erzeugten Game initialisiert. Daraufhin wird die Methode getBestAction() aufgerufen. Der Rückgabewert dieser Methode enthält die Aktion, die von der Software, zu dem Zeitpunkt als am effizientesten eingestuft wird bzw. zum Retten der Welt am zielführendsten ist.

Aktionen werden vom ActionCreater erstellt und dem GameExecutor bereitgestellt. Damit die Software in der Lage ist, zu bewerten, welche Aktion zu einem Zeitpunkt die „beste“ ist, muss sie die Durchführung der einzelnen Aktionen untereinander, anhand eines Bewertungssystems, vergleichen. Außerdem muss die Software wissen, wann die Durchführung von Aktionen sinnvoll ist. Für den Fall, dass zum Beispiel das Spiel bereits gewonnen wurde, ist die Durchführung weiterer Aktionen uneffektiv bzw. nutzlos. Deshalb wird vom ActionCreater zunächst eine Aktion definiert, die das Spiel beendet. Aus dem Output des Spiels wird abgelesen, welchen Status das Spiel zu einem Zeitpunkt hat. Sofern der Status von „pending“ auf „win/loss“ wechselt, ruft der GameExecutor den ActionCreater nicht mehr auf. Solange das Spiel den Status „pending“ hat, liefert der ActionCreater dem GameExecutor bei Abfrage eine Übersicht über sämtliche, durchführbare Aktionen. Eine Aktion verfügt stets über folgende Eigenschaften:

- *Points:* Gibt an, wie viele Punkte die Durchführung dieser Aktion kostet.
- *ThreatIndicator:* Ein kalkulierter Wert, der Auskunft darüber gibt, wie bedrohlich bzw. wichtig die Durchführung dieser Aktion ist. Der Wert beschreibt die Gefahr, die die jeweilige Aktion bewältigen kann. Der ThreatIndicator Wert wird für jede Aktion im ThreatEvaluater berechnet. In die Berechnung fließen auch ThreatIndicator Werte von Pathogenen und Städten mit ein. Diese werden dediziert in der Klasse ThreatIndicator berechnet. Die ThreatIndicator Berechnung von Aktionen basiert zunächst auf die Verrechnung von sämtlichen Komponenten, die durch die Ausführung der Aktion beeinflusst werden. Die Berechnungslogik soll anhand des folgenden Beispiels verdeutlicht werden:

  {"type": "closeAirport", "city": "münchen", rounds": 3}

  Damit der ThreatEvaluator für die oben dargestellte Aktion den ThreatIndicator berechnen kann, schaut er sich zunächst die Ausprägungen der City „München“ an. Die Werte der Ausprägung werden wie folgt ins numerische System übertragen:

Table 1: Werteangaben City ins numerische System

++	0.8
+	0.9
O	1
-	1.1
--	1.2

Diese Werte werden anschließend mit vordefinierten Faktoren multipliziert. Das Einführen der Faktoren hat den Hintergrund, dass sich die einzelnen Eigenschaften einer Stadt unterschiedlich darauf auswirken, wie sehr eine Stadt von dem Befall einer Krankheit betroffen ist. Die Eigenschaften werden wie folgt gewichtet:

Table 2: Gewichtung der City Eigenschaften

Economy	1
Government	1
Hygiene	1.08
Awareness	1.05

Abschließend werden sämtliche Eigenschaftsausprägungen erneut mit einander multipliziert, woraus der City ThreatIndicator resultiert. Die City München hätte demnach mit den Eigenschaften aus Figure 3 einen ThreatIndicator von 0,88.

Table 3: Beispiel Eigenschaftsausprägungen City München

Economy	+

Government	--
Hygiene	++
Awareness	+

Nachdem der ThreatEvaluator den ThreatIndicator der City erhalten hat, prüft er, ob die City von einem „Outbreak-Event“ betroffen ist. Ist dies der Fall, wird für das oder die Pathogene, die in der Stadt existent sind, der ThreatIndicator berechent. Das Berechnungsmuster zum Berechnen des Pathogen ThreatIndicators gleicht dem des City ThreatIndicators. Allerdings werden Werte von Pathogenen anders abgebildet, da bei Pathogenen gilt, desto höherer der Wert, desto Größer die Bedrohung bzw. Gefahr, die von dem Pathogen ausgeht.

Table 4: Werteangaben Pathogene ins numerische System

++	1.2
+	1.1
O	1
-	0.9
--	0.8

Zudem verfügt ein Pathogen über andere Eigenschaften als eine City. Die Gewichtung der Pathogen Eigenschaften ist folgendermaßen definiert:

Table 5: Gewichtung Pathogen Eigenschaften

Infectivity	1.05
Mobility	1
Duration	1
Lethality	1.08

Sei die Stadt München zum Berechnungszeitpunkt von dem Pathogen „Coccus innocuus“ mit den Eigenschaften aus Figure 6 befallen, ergibt sich ein Pathogen ThreatIndicator von 0,99

Table 6: Beispiel Eigenschaftsausprägungen Pathogen Coccus innocuus

Infectivity	+
Mobility	o
Duration	--
Lethality	o

Diese Werte stellen das Grundgerüst des Berechnungsschemata dar. Der ThreatEvaluator nutzt diese Werte unteranderem, um zu evaluieren, welche Gefahr

durch die Schließung des Flughafens reduziert wird. Da die Durchführung jeder Action unterschiedliche Game-Objekte beeinflusst, bedarf die Festlegung des ThreatIndicator Werts einer Action, einem individuellen Berechnungsverfahren. Die Methode „getThreatOfCityAndPathogens“ liefert einen multiplizierten Wert aus City- und Pathogen ThreatIndicator. Wie bereits im Kapitel 2.4 beschrieben, reduziert die Schließung des Flughafens die Geschwindigkeit mit der sich Pathogene verbreiten können. Um den ThreatIndicator zu berechnen, ruft der ThreatEvaluator daher zunächst eine ArrayList von sämtlichen Städten auf, die über einen Direktflug die Stadt München erreichen können. Nun summiert der ThreatEvaluator die „getThreatOfCityAndPathogen“ Werte aller Cities, die Bestandteil der ArrayList sind. Von diesen wird der Durchschnittswert ermittelt und anschließend mit einem vordefinierten Wert faktorisiert. Durch Beobachtungen konnte ermittelt werden, dass bestimmte Actions eine höhere Auswirkung auf die Bekämpfung der Pathogene ausüben. Die Einführung einer zusätzlichen Gewichtungs-Matrix trägt demnach dazu bei, dass gewisse Actions höher gewertet werden als andere. Bei Actions, die sich über mehrere Runden erstrecken können, kann dadurch zudem berechnet und verglichen werden, für wie viele Runden die Aktion durchgeführt werden soll. Die Tabelle 7 stellt dar, wie die einzelnen Actions zusätzlich hoch- bzw. runtergestuft werden.

Table 7: Übersicht individuelle Action Faktoren

BOOST_ROUND_THREAT_PUT_UNDER_QUARANTINE	1.25
BOOST_ROUND_THREAT_CLOSE_AIRPORT	1.12
BOOST_ROUND_THREAT_CLOSE_CONNECTION	1.04467
PREVALENCE_FACTOR_BOOSTER	0.75
END_ROUND	1.2
DEVELOP_MEDICATION	1.4
DEPLOY_MEDICATION	0.75
PUT_UNDER_QUARANTINE	0.98
CLOSE_AIRPORT	0.9
CLOSE_CONNECTION	0.72
DEVELOP_VACCINE	1.6
DEPLOY_VACCINE	0.6
EXERT_INFLUENCE	0.78
CALL_ELECTION	0.77
HYGIENIC_MEASURES	0.787
LAUNCH_CAMPAING	0.775

Die Implementierung von diesem Lösungsansatz bietet den Vorteil, dass Actions basierend auf den Beobachtungen gewichtet werden können.

Nachdem der GameEvaluator die Gesamtübersicht aller durchführbaren Aktionen erhalten hat, ist es die Aufgabe des DecisionMakers, die Auswahl der Aktion zu treffen. Dabei wählt der DecisionMaker zunächst die erste Action aus der ArrayList aus. Anschließend wird der ThreatIndicator Wert von dieser Aktion mit den ThreatIndicator Werten aller anderen Aktionen verglichen. Damit der Vergleich adäquat erfolgen kann, müssen auch die Kosten berücksichtigt werden, die die Durchführung einer Aktion verursacht. Um den Unterschied der Punktwerte nicht außer Acht zu lassen, wird der ThreatIndicator Wert der Aktion mit dem geringeren Punktwert erhöht. Dazu wird der ThreatIndicator Wert mit einem Faktor für die Punktdifferenz multipliziert. Der Faktor für eine Punktdifferenz von einem Punkt (1,0103) wird dabei mit der Punktdifferenz exponiert, um den Faktor für die vorhandene Punktdifferenz zu erhalten.

3 Auswertung

Um die Faktoren anzupassen bzw. zu ermitteln, ob die veränderten Faktorenwerte einen positiven Einfluss auf die Spiellogik ausüben, wurden die Eigenschaften einer Runde zwischengespeichert. Dadurch wurde bewerkstelligt, dass Änderungen an der Spiellogik mit einander verglichen werden konnten, da Beobachtungen auf Basis des gleichen Spiels erfolgen konnten.

Zum Ermitteln der Zuverlässigkeit, wurden insgesamt 255 Spiele ausgeführt und beobachtet. Dabei konnte die Software 73 Spiele mit dem Outcome: „win“ beenden bzw. gewinnen. Dies entspricht einer Win-Rate von ca. 28,63%. Es wurde festgestellt, dass bei den Runden, die nicht gewonnen werden konnten, die Rundenanzahl im Mittel bei ca. 37 lag. Bei den 73 malen, in den die Software die Pandemie bezwingen konnte, befand sich das Spiel im Durchschnitt in der 47. Runde. Außerdem wurde das Spiel mit einer Wahrscheinlichkeit von 34,44% gewonnen, sobald das Spiel Runde 24 überwunden hat.

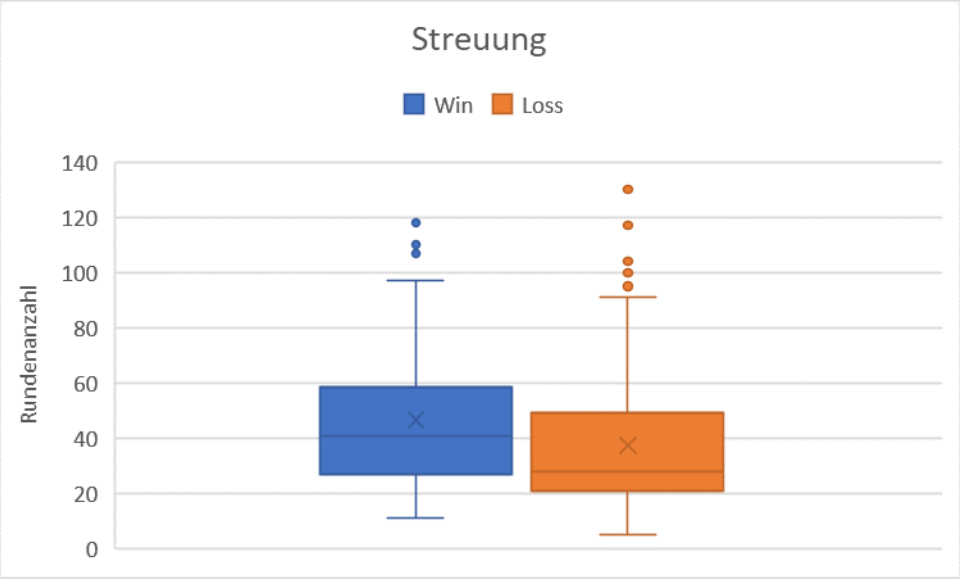

Abbildung 1: Streuung der Ergebnisse

4 Diskussion

Der unter Kapitel 2.5 vorgestellte Lösungsansatz zum Berechnungsmodell des ThreatIndicator war umstritten. Da das Berechnungsmodell die Kernlogik der Software wiederspiegelt, hat sich die Gruppe im Vorfeld der Umsetzung intensiv darüber beraten, ob ein Machine-Learning Verfahren eingesetzt werden soll, welches zusätzlich, neben den ThreatIndicator, einen Einfluss auf die Entscheidungsfindung ausübt. Dieser Ansatz beruhte darauf, einen Deep-Learning Algorithmus zu implementieren, der berücksichtigt, wie sich Entscheidungen der vergangenen Runden auf den Spielfluss ausgewirkt haben. Die Intention dahinter verfolgte das Ziel, dass die Software dazu imstande seien sollte, unabhängig von der implementierten Berechnungslogik, Kennzahlen einfließen zu lassen, die von der Gruppe im Zuge der Entwicklung nicht berücksichtig wurden oder von dem Berechnungsverfahren nicht verwendet werden. Vergleichbare Verfahren werden bereits in modernen Kliniken und Krankhäusern zum Diagnostizieren von Krankheitsbildern genutzt, da der Deep-Learning Ansatz dazu imstande ist, Informationen in seine Bewertungen einfließen zu lassen, die der Mensch tendenziell vernachlässigt. (Köneke, 2018) Des Weiteren bietet das Machine-Learning Verfahren den Mehrwert, dass zielgerecht und angemessen auf unbekannte Objekte reagiert werden kann. (Ionos, 2019) Dieser Ansatz wurde allerdings verworfen, da die Aufgabenstellung definiert, dass die Software zustandlos sein muss. Als dieser Aspekt zur Sprache kam, waren sich die Gruppenmitglieder allesamt darüber einig, dass die Zustandslosigkeit mit der Einführung eines Machine-Learning Algorithmus nicht mehr gewährleistet werden kann.

In einer fortgeschritteneren Phase des Projekts wurde festgestellt, dass die Software dazu neigte, Aktionen nach einem ungewollten, nicht definierten Muster, auszusuchen. Dieses Ereignis war auf Berechnungen des ThreatEvaluators zurückzuführen. Einzelne Aktionen wurden aus Basis einer Fehlkalkulation zu stark gewichtet, wodurch andere Aktionen kaum bis gar nicht mehr ausgeführt wurden. Ein Ansatz zum Lösen des Problems beruhte auf die Einführung eines Counters, welcher die Ausführung der Aktionen speichern sollte, um die Durchführungs-Häufigkeit von Aktionen bei der Auswahl der Aktion zu berücksichtigen. Allerdings wurde dieser Ansatz ebenfalls auf Grund der oben genannten Thematik nicht umgesetzt, da durch die Implementierung des Counters die Zustandslosigkeit nicht mehr gewährleistet wäre.

Zwar bietet die bereitgestellte Lösungslogik, wie unter Kapitel 2.5 beschrieben, die Möglichkeit, Faktoren basierend auf Beobachtungen anzupassen, allerdings wird durch diesen Lösungsansatz nicht gewährleistet, dass die vorliegende Justierung der Faktoren die effizienteste ist. Es ist denkbar, dass die Software effizienter reguliert werden könnte, wenn ein längerer Beobachtungszeitraum vorhanden wäre, in dem das Verhalten des Spiels analysiert werden würde.

5 Softwarearchitektur

5.1 Grundaufbau

Die Softwarearchitektur stützt sich auf die Verwendung von abstrakten Klassen. Dabei wird basierend auf JAVA rein objektorientiert programmiert. Im Zuge der Entwicklung lag der Fokus hauptsächlich darauf, wie bewerkstelligt wird, dass die entwickelte Software in der Lage ist, Aktionen untereinander zu vergleichen und deterministisch eine als besser zu bewerten.

5.2 Wartbarkeit

Im Rahmen von Softwareprojekten kann es während der Betriebsphase dazu kommen, dass durch neue Anforderungen oder gefundenen Fehlern eine Anpassung an die Software notwendig wird. Bei einer gut wartbaren Software kann man mit Änderungen einfacher und günstiger umgehen. Zudem wird die Anzahl der Änderungen möglichst klein gehalten. (Meiert, 2019). Während der Softwareentwicklung wurde stets darauf geachtet, dass die Software leicht zu erweitern ist. Ein Szenario, in dem die Modularität gefordert wird, ist das Hinzufügen einer neuen Aktion. Unterschiedliche Spiel-Klassen, wie Events, Aktionen und Pathogene erben ihre Eigenschaften von übergeordneten, abstrakten Klassen. Demnach wäre nicht weiter komplex, das Spiel zu erweitern. Wenn zum Beispiel ein neues Event ergänzt werden muss, muss lediglich eine Klasse mit der Eigenschaft „extends Event“ im Package erstellt werden und in der EventFactory Klasse als neuer Fall hinzugefügt werden. Sofern es sich um ein Event handelt, welches direkten Einfluss auf andere Game-Objekte wie Aktionen hat, muss in der jeweiligen Methode des ActionCreators bzw. ThreatEvaluator das neue Event verarbeitet werden. Für eine neue Aktion muss, ähnlich wie bei neuen Events, zunächst eine neue Klasse, mit der Eigenschaft „extends Action“, im Action Package erstellt werden. Um für die Aktion ein JSON Objekt bereitstellen zu können, wird beim JsonActionProvider eine neue Methode benötigt, die für die neue Aktion ein JsonObject zurückgibt. Dies ist beim ThreatEvaluator ähnlich. Hier ist ebenfalls eine neue Methode zu schreiben, die den ThreatIndicator Wert der Aktion zurückliefet. Ferner muss der ActionCreator um eine Methode ergänzt werden, die bei einem Spielzustand sämtliche durchführbare Aktionen des neu hinzugefügten Aktiontyps zurückgibt. Die Methode ist anschließend in der Methode getAllPossibleActions aufzurufen. Für Anpassungen an der Gewichtung der Events bzw. Bewertung der Aktionen sind an mehreren Stellen Stellschrauben (Konstante Faktoren) vorhanden, die die Entscheidung, die die Software trifft, maßgeblich beeinflussen. Der Aufwand für Anpassungen an der Gewichtung ist dementsprechend gering. Die Auslagerung der JsonObejekte in den JsonActionProvidern und die Berechnung des threatIndicator Werts in den ThreatEvaluator wurde bewusst so gewählt, um Aktions-Klassen möglichst einfach und übersichtlich zu halten, sodass Aufgaben der vorhandenen Klassen klar definiert sind. Diese Umsetzung bietet zudem die oben beschriebene Möglichkeit, gewisse Bereiche der Software zu abstrahieren. Es ist, um sich zunächst einen Überblick über die Software zu machen, nicht von Nöten, die Action Klasse oder das Berechnungsmodell des ThreatIndicators zu verstehen, sondern zunächst wichtig, zu erkennen, dass es diese Werte gibt und was sie für Auswirkungen auf den weiteren Programmablauf haben.

5.3 Test

Das Konzept zum Testen der Software überprüft primär, ob die Logik hinter der Auswahl der Action funktioniert. Hierfür wurden einige JUnit Tests angelegt. In diesen werden unter anderem ThreatIndicator Werte auf ihre Richtigkeit getestet. Darüber hinaus wird überprüft, ob der DecisionMaker beim Vergleich zweier Actions die Kosten der beiden Actions in den ThreatIndicator Wert ordnungsgemäß berücksichtigt.

Für einige Testfälle ist es notwendig, Mock-Objekte zu erstellen, da das Erstellen realer Objekte zu aufwändig wäre. Des Weiteren können durch die Erzeugung von Mock-Objekten eine einzelne Klasse abgesondert von anderen Umständen, wie zum Beispiel Fehlern, in anderen Klassen betrachtet werden. Zudem vereinfacht diese Umsetzung das Testen besonderer Konstellationen. Zum Beispiel der Test im DecisionMaker getBestActionFromTwoActionsWithEqualThreatIndicatorsTest(). Zwei Aktionen mit demselben ThreatIndicator Wert bei unterschiedlich großen Punktewerten zu erstellen wäre durch alleiniges Erstellen eines Game Objektes nur sehr schwierig bis gar nicht möglich. Dieser Testfall kann mit Mock-Objekten sehr leicht durchgeführt werden. Es werden Objekte mit der Funktion mock(CLASS.class) erstellt (CLASS ist hier eine Action z.B. closeAirportAction) und das Verhalten beim Ausführen bestimmter Methoden definiert (z.B.: when(closeAirportAction.getPoints()).thenReturn(20);). Somit kann auch ein Verhalten in Sonderfällen getestet werden.

Bei herkömmlichen Testfällen werden JsonObjects erstellt, um aus diesen wiederum GameObjects zu erstellen. Dies ist zum Beispiel bei dem CityFactoryTest der Fall. Bei diesem ist ist es notwendig, die Korrektheit des erstellten City Objektes zu überprüfen. Da die CityFactory dafür ein JsonObject benötigt, wird dieses in der @Before Methode (setUp()) erstellt.

5.4 Konventionen

Da es sich bei diesem Projekt um eine Gruppenarbeit handelt, ist die Festlegung von Gruppenkonventionen von essenzieller Bedeutung gewesen, da jeder Entwickler seine persönlichen Präferenzen besitzt. Die Gruppenmitglieder haben sich daher primär auf Standard Java Konventionen geeinigt, die unter anderem im Rahmen des Studiums in dem Modul „Praxis der Softwareentwicklung“ von dem Dozenten Lennart Gamradt vermittelt wurden.

6 Fazit

Im Rahmen des Projekts wurde eine Softwareanwendung entwickelt, die in der Lage ist, unterschiedliche Spieleigenschaften zu verrechnen, um Actions auszuwählen, die dazu beitragen, dass die fiktive Welt vor der Eliminierung der menschlichen Spezies bewahrt wird. Jedoch ist zu beachten, dass bei der Auswahl der Aktionen Verbesserungspotenzial bestehen könnte.

Literaturverzeichnis

Fritz, M. (07. Februar 2008). *MMLOGISTIK.* Von https://www.mm-logistik.vogel.de/modularer-aufbau-erhoeht-stabilitaet-von-software-zur-lagerverwaltung-a-197325/ abgerufen

Ionos. (31. Juli 2019). *Ionos.* Von https://www.ionos.de/digitalguide/online-marketing/web-analyse/was-ist-machine-learning-so-lernen-maschinen-denken/ abgerufen

Köneke, V. (13. August 2018). *Zeit Online.* Von https://www.zeit.de/digital/internet/2018-08/deep-learning-medizin-kuenstliche-intelligenz-neurologie-augenheilkunde abgerufen

Meiert, J. O. (02. Februar 2019). *Meiert.com.* Von https://meiert.com/de/publications/articles/20090907/ abgerufen