Informatik Cup 2020 Pandemie

Theoretische Ausarbeitung

des Teams

Elemental Discharge

bestehend aus

Nils Braun, Rouven Anderer und Lukas Göbl

15.01.2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

KI	**Künstliche Intelligenz**
API	**Application Programming Interface**
UI	**User Interface**

1. Lösungsansatz

Das gegebene Spiel lässt offen, mit welcher Technologie eine Lösung ermittelt werden sollte. Es stellt sich die Frage, ob parametrisierte oder lernende Algorithmen sich sinnvoll auf das Problem anwenden lassen oder ein prozeduraler Algorithmus notwendig ist. Im Folgenden wird die Möglichkeit der Anwendung von Künstlicher Intelligenz unter den fünf Gesichtspunkten nach Michael Chui, James Manyika und Mehdi Miremade in ihrem Artikel "*What AI can and can't do (yet) for your business*" bewertet [1].

1.1 Daten Labeling

Das meist verwendete Modell von KI ist das überwachte Lernen. Dies bedeutet, dass ein Mensch die Datensätze, die für das Trainieren der KI verwendet werden, selbst beschriften und kategorisieren muss. Im Falle des gegebenen Spiels wäre dieses nur unter großem Aufwand realisierbar. Der Spieler müsste über umfassende Kenntnisse über das Spiel verfügen, um in jeder Situation eine Aktion in ihrer Güte bewerten zu können. Dies wäre allerdings notwendig, um ein sinnreiches Datenlabeling zu gewährleisten. Daher ist das Daten Labeling beim überwachten Lernen in dem Beispiel des gegebenen Spiels sehr fehleranfällig.

Eine Lösung für dieses Problem könnte das verstärkte Lernen sein. Diese Methode lernt Aufgaben nach dem einfachen *Trial-and-Error* Prinzip. Dabei wird die KI für sinnvolle Aktionen belohnt und für weniger sinnvolle Aktionen bestraft [2]. Dies bedingt eine Bewertungsfunktion. Die Bewertung würde dabei entweder zum Ende des Spiels aus den Ergebnissen im Hinblick auf Ausgang und benötigte Rundenanzahl greifen, oder aber auf Basis einzelner Runden oder Aktionen. Unter Berücksichtigung der nahezu unendlich vielen Möglichkeiten, wie ein einzelnes Spiel abhängig von Aktionen verlaufen kann, bietet ersteres der KI kaum die Möglichkeit auf die Sinnhaftigkeit einzelner durchgeführter Aktionen zurückzuschließen. Von einem Training auf Basis einzelner Runden oder Aktionen ließen sich zwar sinnvolle Lernfortschritte der KI erwarten, allerdings bedingt dies eine Bewertung einzelner Aktionen oder Runden. Dies lässt sich nur schwierig und mit umfangreichem Wissen über die Funktionsweise des Problems bewerkstelligen und ist somit ebenfalls unpraktikabel.

1.2 Erhalten massiver Trainings Datensätze

Um eine KI ausreichend zu trainieren ist eine große Anzahl an Datensätzen notwendig. Im Falle des gegebenen Spiels stellt das Erhalten von Datensätzen kein Problem dar, da das Spiel schnell und häufig ausgeführt werden kann.

1.3 Das Erklärungsproblem

Das Erklärungsproblem beschreibt das Problem, dass das Ergebnis der KI nicht nachvollzogen werden kann. Dies trifft auf eine KI, die das gegebene Spiel spielen würde, ebenfalls zu, da sie sich bei einer bestimmten Entscheidung möglicherweise nur auf einen kleinen Teil des gegebenen Spielstandes bezieht. Diese Tatsache stellt in diesem Zusammenhang ein erhebliches Problem dar. Bestimmte Aktionen können bei verschiedenen Spielsituationen sehr unterschiedliche Folgen haben. Wenn es dem Entwickler nicht möglich ist, eine bestimmte Entscheidung der KI nachzuvollziehen, da diese möglicherweise auf unerwarteter Basis getroffen wurde, ist eine gezielte Verbesserung der KI nahezu unmöglich.

1.4 Generalisierbarkeit des Lernens

Mit der Generalisierbarkeit des Lernens ist gemeint, wie gut die KI gelerntes Wissen auf eine neue Situation anwenden kann. Wenn sich die neue Situation zu stark vom gelernten Wissen unterscheidet, kann die KI keine gute Lösung geben. Im gegebenen Spiel können sich unterschiedliche Szenarien stark voneinander unterscheiden. Dies folgt aus der Vielzahl unterschiedlicher Pathogene, unterschiedlichen Städte, die am Anfang des Spiels infiziert sind und unterschiedlichen Ereignissen, die im Laufe des Spiels auftreten können. Entsprechend schwierig ist es, die gesamte Komplexität des Spiels in einer Datenmenge zu repräsentieren und der KI somit eine sinnvolle Lerngrundlage zu schaffen.

1.5 Fazit

Gerade die Komplexität und die teilweise gravierende und vor allem unterschiedliche Auswirkung verschiedener Aktionen bei ähnlichem Spielstand macht es äußerst schwierig, aus den Daten des Spiels eine repräsentative Datenmenge zu extrahieren, auf deren Basis eine KI trainiert werden könnte. Insbesondere im Hinblick auf Nachvollziehbarkeit und individuelle Anpassbarkeit stellt ein prozeduraler Algorithmus daher die bessere Alternative dar.

2. Reverse Engineering

Entscheidend für das Verstehen des Spiels und das Umsetzen eines automatisch lösenden Algorithmus sind Erkenntnisse über die zugrunde liegende Funktionsweise des Spiels. Da diese nicht bekannt sind, müssen über so genanntes Reverse Engineering, also dem gezielten Untersuchen von Strukturen, Zuständen und Verhaltensweisen, Rückschlüsse gezogen werden.

2.1 Grundlagen

Die grundlegenden Mechaniken sind durch einfache Untersuchungen und ohne wahrscheinlichkeitsbasierte Rückschlüsse zu erkennen und im Folgenden aufgelistet.

2.1.1 Städte

Im Spiel gibt es 260 verschiedene Städte. Jede dieser Städte besitzt folgende Eigenschaften:

- Stadtname
- Bevölkerung
- *Hygiene*
- *Government*
- *Awareness*
- *Economy*

Hygiene, *Government*, *Awareness* und *Economy* werden mit einer Bewertung von “--” bis “++” angegeben. Alle Städte, mit wenigen Ausnahmen, sind mit Flugverbindungen untereinander verbunden.

2.1.2 Pathogene

Pathogene können die Bevölkerung einer Stadt infizieren und über den Verlauf mehrerer Runden einen Teil der Infizierten töten. Ein Pathogen verfügt über folgende Eigenschaften:

- *Infectivity*
- *Lethality*
- *Mobility*
- *Duration*

Infectivity, *Lethality*, *Mobility* und *Duration* werden mit einer Bewertung von "--" bis "++" angegeben.

2.1.3 Events

Events beschreiben bestimmte Ereignisse im Spiel. Diese können global gelten, so kann zum Beispiel ein Heilmittel für ein Pathogen in Entwicklung sein, oder lokal in einzelnen Städten gelten, beispielsweise wenn eine Stadt von einem Pathogen infiziert wurde. Zusätzlich lassen sich Events auch in permanente und nicht permanente Events aufteilen.

2.1.4 Spielablauf

Das Spiel beginnt mit ein bis drei Pathogenen, die jeweils eine oder zwei Städte infiziert haben. Diese breiten sich anschließend hauptsächlich über Flugverbindungen, aber auch ohne diese, aus. In den verschiedenen Städten infizieren die Pathogene die Bevölkerung und töten einen Teil dieser. Dabei kann eine Stadt immer nur von einem Pathogen befallen sein. Dieser Vorgang wiederholt sich bis entweder alle Pathogene ausgestorben sind oder die Weltbevölkerung unter 50% sinkt, woraus das Gewinnen bzw. das Verlieren des Spiels folgt.

2.2 Fortgeschrittene Rückschlüsse

Tiefergehende Rückschlüsse lassen sich nur auf Basis von umfangreichen Datenmengen ziehen. Je größer und repräsentativer die Datenmenge, desto genauer und sicherer sind dabei die Erkenntnisse. Es ist allerdings wichtig zu beachten, dass die Rückschlüsse dennoch wahrscheinlichkeitsbasiert sind und nur Vermutungen darstellen.

2.2.1 Verbreitung eines Pathogens

Der erste denkbare Lösungsansatz, das Spiel zu gewinnen, ist es zu versuchen die Verbreitung der Pathogene einzudämmen. Hierfür werden dem Spieler drei Aktionen bereitgestellt:

Tabelle 1 Spieleraktionen zum Eindämmen eines Pathogens

Aktionsname	Effekt
closeConnection	Schließt eine Flugverbindung zwischen zwei Städten für eine beliebige Anzahl an Runden
closeAirport	Schließt den Flughafen einer Stadt und somit alle ausgehenden Flugverbindungen für eine beliebige Anzahl an Runden
putUnderQuarantine	Setzt eine Stadt für eine beliebige Anzahl an Runden unter Quarantäne. Die Quarantäne versichert, dass ein Pathogen, das sich der Stadt befindet, sich nicht von dort aus ausbreiten kann. Zusätzlich können keine neuen Pathogene die Stadt infizieren

Da *closeConnection* sozusagen nur die schwächere Variante von *closeAirport* ist, wird im Folgenden nur *closeAirport* betrachtet. Im folgenden Experiment wird versucht, die Verbreitung eines Pathogens in der ersten Runde zu verhindern. Dafür wird von der Stadt, in der das Pathogen startet, einmal der Flughafen gesperrt und einmal die Stadt unter Quarantäne gesetzt.

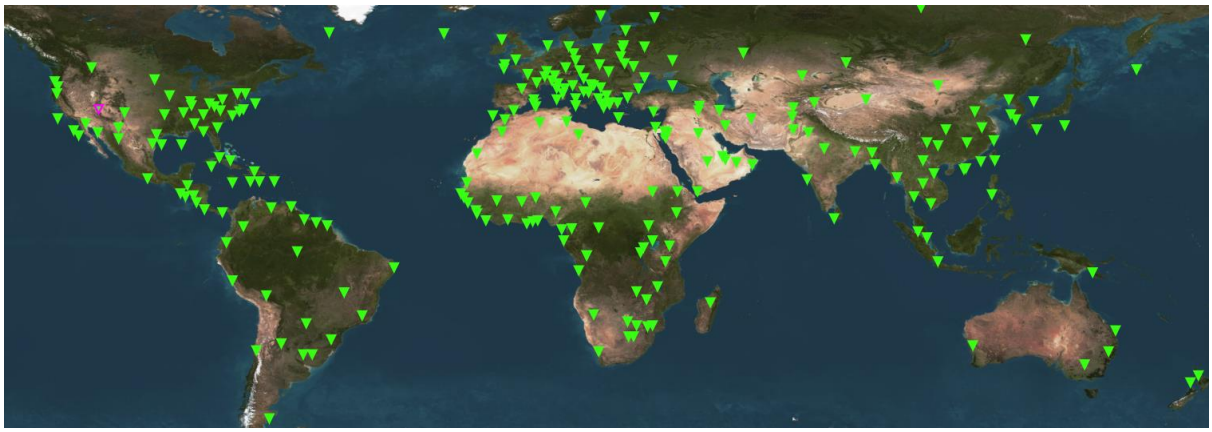

Abbildung 1 Erste Runde eines Spiels

In Abbildung 1 Erste Runde eines Spiels wird die erste Runde des Spiels dargestellt. Es existieren zwei Pathogene. Das erste startet in Nordamerika und wird pink dargestellt. Das zweite Pathogen startet in Ägypten und wird dunkelgrün dargestellt.

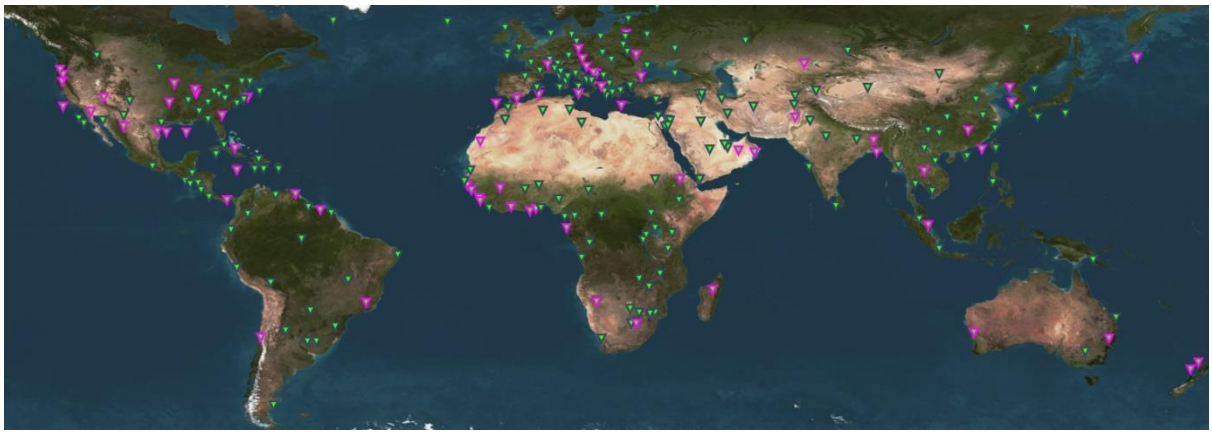

Abbildung 2 Zweite Runde (ohne Aktion)

Ohne eingreifen des Spielers infiziert das pinke Pathogen in der zweiten Runde ca. 40 % der Welt. Der Rest ist von dem dunkelgrünen Pathogenen infiziert (siehe Abbildung 2 Zweite Runde (ohne Aktion)).

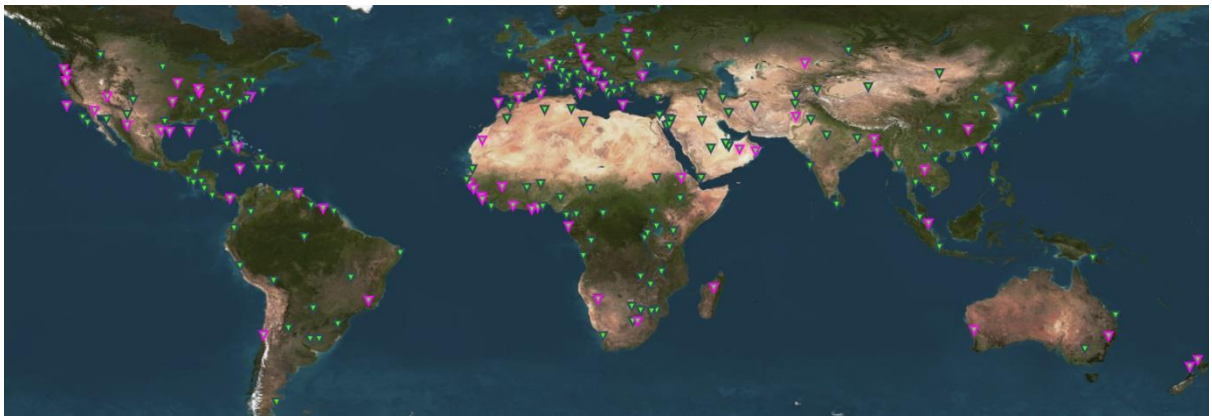

Abbildung 3 Zweite Runde (geschlossener Flughafen)

Mit geschlossenem Flughafen infiziert das Pathogen immer noch ca. 40 % der Welt. Das Schließen des Flughafens der infizierten Stadt verhindert das Verbreiten des Pathogens, wenn überhaupt, nur geringfügig (siehe Abbildung 3 Zweite Runde (geschlossener Flughafen)).

Abbildung 4 Zweite Runde (Quarantäne)

Wie in Abbildung 4 Zweite Runde (Quarantäne) zu sehen ist, verbreitet sich ein Pathogen nicht aus der Stadt, in der es startet, wenn diese unter Quarantäne gesetzt wird.

Aus diesem Experiment lassen sich folgende Schlussfolgerungen ziehen:

1. Pathogene verbreiten sich nicht nur über Fluglinien, sondern wahrscheinlich auch über Land.
2. Eine infizierte Stadt in der ersten Runde unter Quarantäne zu setzen verhindert das Verbreiten des Pathogens in der nächsten Runde. Zusätzlich verringert dies die Verbreitung in den folgenden Runden, da die restlichen Pathogene bereits in der zweiten Runde einen Großteil, wenn nicht sogar alle, weiteren Städte der Welt infiziert haben.
3. Da *putUnderQuarantine* vergleichsweise viele Punkte kostet und *closeAirport* bzw. *closeConnection* keinen großen Einfluss hat, ist das Verhindern der Verbreitung eines Pathogens, das bereits mehrere Städte infiziert hat, vergleichsweise schwer.

2.2.2 Verhalten eines Pathogens in einer Stadt

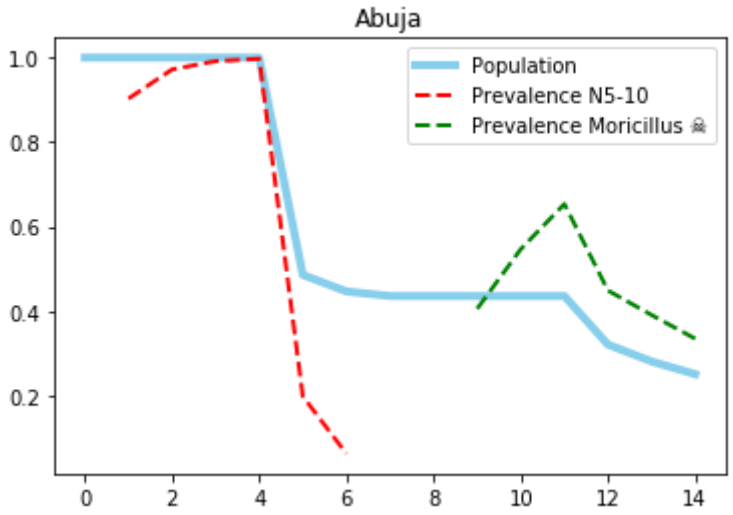

Abbildung 5 Werdegang Abuja

In Abbildung 5 Werdegang Abuja ist die Entwicklung der Einwohnerzahl einer Stadt über ein Spiel hinweg abgebildet. Die blaue Linie beschreibt die momentane Bevölkerung der Stadt in Prozent, wobei 100% die Bevölkerung ist, welche die Stadt am Anfang des Spiels hatte. Die rote bzw. grüne Linie beschreibt wieviel Prozent der zur auf der x-Achse angegebenen Runde Bevölkerung der Stadt ein Pathogen infiziert hat. In den Runden, in denen keine gestrichelte Linie eingezeichnet ist, ist die Stadt nicht von einem Pathogen infiziert.

Aus dem Vergleichen mehrere Städte und deren Werdegang über das Spiel, lassen sich folgende Beobachtungen machen.

- Wenn ein Pathogen eine Stadt infiziert, infiziert es direkt einen gewissen Prozentsatz der Bevölkerung
- Über den Verlauf der nächsten Runden infiziert das Pathogen immer mehr Menschen, bis es ein Maximum erreicht
- In den Runden nach dem Erreichen des Maximums beginnt das Pathogen an einen Teil der Infizierten zu töten

Aus diesen Beobachtungen lassen sich bereits folgende Schlussfolgerungen ziehen:

- Das Verteilen von Heilmitteln ist am effektivsten, wenn es in der Runde angwendet wird, in der das Pathogen sein Maximum an Infizierten erreicht.
- Das Verteilen von Impfungen ist am effektivsten, wenn die Stadt entweder gerade erst infiziert wurde oder noch nicht infiziert ist.
- Die Eigenschaft *Mobility* eines Pathogen gibt an, wie stark bzw. wie schnell es sich verbreitet.

2.2.3 Verhalten Pathogen in verschiedenen Städten

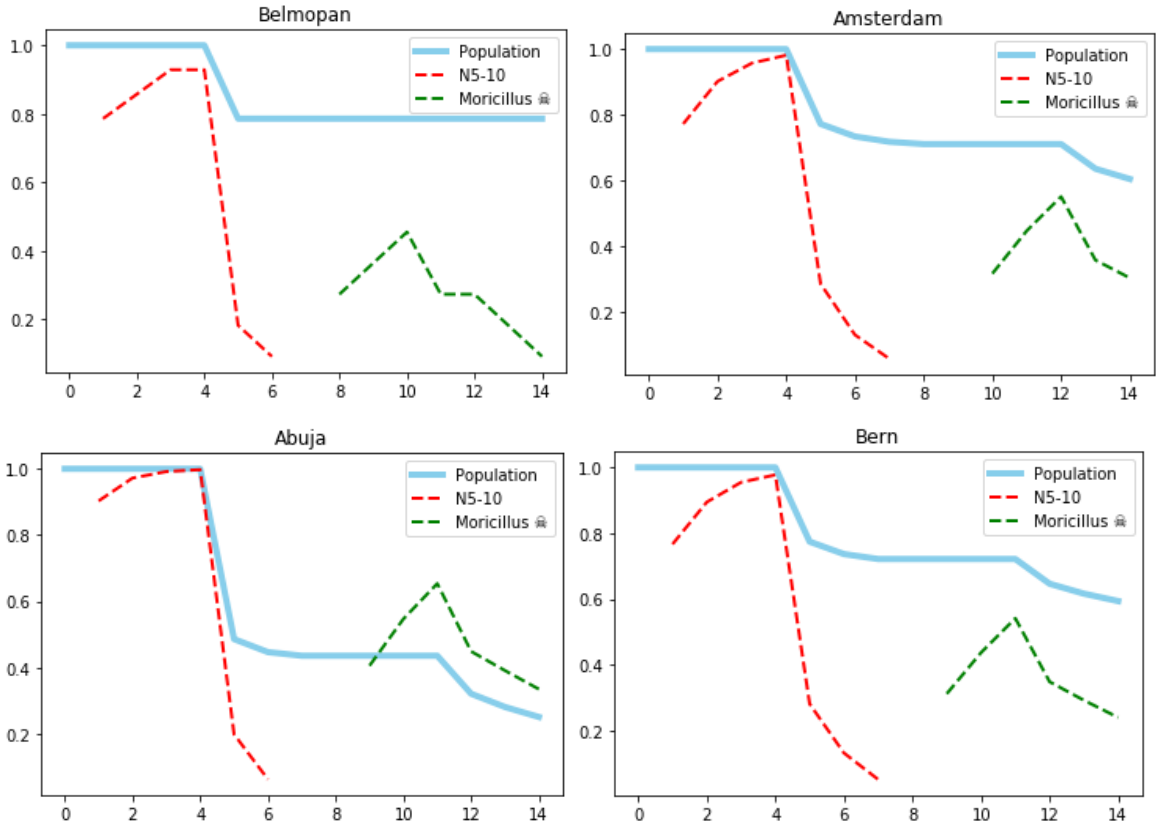

Abbildung 6 Werdegang des Pathogen N5-10 in verschiedenen Städten

Durch das Analysieren des Werdegangs verschiedener Städte in mehreren Spielen (siehe Abbildung 6 Werdegang des Pathogen N5-10 in verschiedenen Städten), die vom selben Pathogen infiziert wurden, lassen sich folgende Aussagen treffen:

Die Anzahl an Runden, die dasselbe Pathogen benötigt, um das Maximum an Infizierten zu erreichen, ist in jeder Stadt gleich.

Wie viel Prozent der Bevölkerung am Anfang infiziert werden, wie viel Prozent am Maximum infiziert sind und wie viel Prozent der Infizierten sterben, sind von Stadt zu Stadt unterschiedlich. Daher liegt es nahe, dass die Eigenschaften der Städte diese Vorgänge beeinflussen. Sind die Eigenschaften allgemein hoch, sterben bspw. weniger Infizierte. Sind die Eigenschaften allgemein niedriger, werden bspw. mehr Prozent der Bevölkerung infiziert.

2.2.4 Verhalten einer Stadt bei verschieden Pathogenen

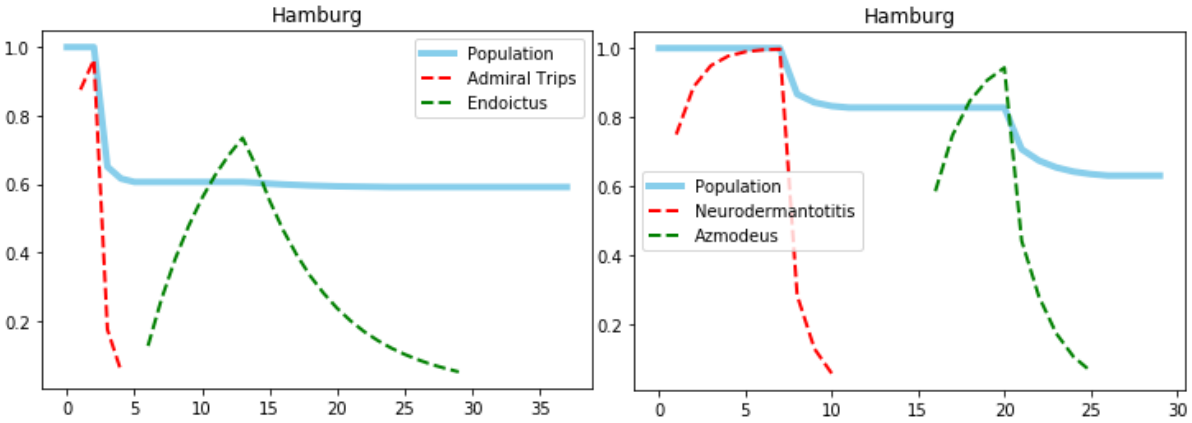

Abbildung 7 Hamburg bei verschieden Pathogenen

Durch das Analysieren des Werdegangs einer Stadt in mehreren Spielen, die von verschiedenen Pathogenen infiziert wurde (siehe Abbildung 7 Hamburg bei verschieden Pathogenen), lässt sich die Bedeutung der Eigenschaften von Pathogen schlussfolgern:

1. *Infectivity* gibt an, wie viel Prozent der Bevölkerung zu Beginn in einer neu befallenen Stadt infiziert sind
2. *Lethality* gibt an, wie viel Prozent der infizierten sterben
3. *Duration* gibt grob an, wie viele Runden das Pathogen in der Stadt bleibt
4. *Dutation* gibt darüber hinaus an, wie viele Runden das Pathogen bis zur Maximalprozentzahl an infizierten Menschen in einer Stadt benötigt

2.2.5 Besondere Events

Besondere Events sind Events, deren Ursache nicht direkt ersichtlich sind. Sie folgen nicht auf Aktionen des Spielers (Aktion “Heilmittel erforschen” ergibt das Event “Heilmittel wird erforscht”), sondern die Ursache ist aus dem allgemeinen Spielstand bzw. den vergangenen Spielständen ableitbar.

Bio-Terrorismus

Bio-Terrorismus ist ein permanentes Event, das in einer Stadt auftreten kann. Es bewirkt, dass sich von der Stadt ein im Spiel bereits vorgekommenes oder ein neues Pathogen ausbreiten kann. Die Stadt selbst ist nicht notwendigerweise direkt selber infiziert. Mögliche Ursachen für den Bio-Terrorismus können sein:

- Viele Städte sind in einem kurzen Zeitrahmen stark mit Pathogenen infiziert
- Wenn in den vergangenen Runden Pathogene häufiger ihr Maximum an infizierter Bevölkerung erreicht haben

In beiden Fällen wäre eine mögliche Prävention des Events das Verteilen eines Heilmittels, trotz niedrigerer Effizienz, bereits vor dem Erreichen des Maximums zu verteilen. Eine mögliche Strategie zur Bekämpfung von einem aufgetretenen Bio-Terrorismus Events wäre, die betroffene Stadt zunächst unter Quarantäne zu setzen, sodass sich das Pathogen nicht verbreiten kann. Gleichzeitig sollte, falls noch nicht vorhanden, ein Heilmittel für das Pathogen entwickelt werden. Sobald das Heilmittel verfügbar ist, sollte der Fokus des Spielers darauf liegen, die betroffene Pathogen in der betroffenen Stadt zu heilen

.

Anti Vaccinationism

AntiVaccinationism ist ein lokales Event, das in einer Stadt auftreten kann, wenn viele Impfungen verteilt wurden. Es schränkt das weitere Verteilen von Impfungen in der Stadt ein.

Weitere Events

Zusätzlich gibt es noch die Events *EconomicCrisis*, *Uprising* und *LargeScalePanic*, die Eigenschaften einer Stadt ändern können.

3. Applikationsstruktur

Im Folgenden wird der Aufbau der Applikation und insbesondere die Aufteilung dieser in Teilapplikationen betrachtet.

3.1 Bestandteile

Die Applikation lässt sich in vier Bestandteile aufteilen.

3.1.1 Game-Info-API

Die Game-Info-API extrahiert Informationen aus dem JSON, das den Spielstand wiedergibt, und formatiert diese. Dadurch werden andere Zugriffspfade auf die Daten geschaffen, um Redundanzen in den Auswertungen zu vermeiden. Die Ergebnisstruktur teilt sich in *Basic Game Information*, die unveränderte Daten wie Rundenzahl, Punkte, etc. beinhalten, und *Extracted Game Information*, die neue Zugriffspfade schaffen. Zum Beispiel werden alle Pathogene, die im Spiel aufgetreten sind, ausgelesen und in ein Objekt *Pathogen With Cities* zusammengefasst. Wie der Name schon sagt, werden zusätzlich alle Städte, die vom entsprechenden Pathogen befallen sind, abgespeichert. Diese Auswertung wird in den Algorithmen zur automatischen Lösung häufig benötigt. Die Daten werden hierbei allerdings nur anders dargestellt, es gibt keine Veränderungen, Wahrscheinlichkeit bedingte Aussagen oder Ähnliches.

3.1.2 Game-API

Die Game-API beinhaltet die Algorithmen für das automatische Lösen des Spiels. Sie stellt die zentrale Teilapplikation zur Lösung des Problems dar. Aufgrund des Umfangs wird die exakte Funktionsweise der Game-API in einem eigenen Kapitel unter 4. Game-API.

Technologisch gesehen sind die Game-API und die Game-Info-API nur zwei verschiedene Endpunkte derselben API. Dies hat den Hintergrund, dass es hauptsächlich die Game-API ist, die sehr häufig auf die Game-Info-API zugreifen muss. Durch das Zusammenlegen in eine API kann die Game-API ohne den Umweg über http-Aufrufe auf einen anderen Server auf den Code zugreifen, wodurch sich die Gesamtgeschwindigkeit des Systems erhöht. Aufgrund der sehr unterschiedlichen Aufgaben und der vorhandenen logischen Trennung wird im Folgenden dennoch von zwei verschiedenen APIs gesprochen.

3.1.3 Web-Frontend

Das Web-Frontend bietet eine Visualisierung des Spielgeschehens und ermöglicht das manuelle Spielen. Die genaue Funktionsweise, Bestandteile und Hintergründe des Designs werden aufgrund des Umfangs in einem eigenen Kapitel unter 5. Visualisierung genauer erläutert.

3.1.4 Verteilungs-API

Die Verteilungs-API stellt eine Schnittstelle zwischen Web-Frontend und der game.exe dar. Sie nimmt die API-Aufrufe der game.exe entgegen und speichert den Spielstatus zwischen. Dieser kann dadurch vom Web-Frontend abgerufen und angezeigt werden. Die Verteilungs-API wartet anschließend auf einen Aufruf, der eine Aktion beinhaltet, um diese an die game.exe weiterzugeben.

3.2 Spielmodi

Die Bestandteile der Applikation werden je nach gewähltem Spielmodus, aufgeteilt in manuell, automatisch und automatisch mit Visualisierung, unterschiedlich miteinander verknüpft.

3.2.1 Automatischer Spielmodus

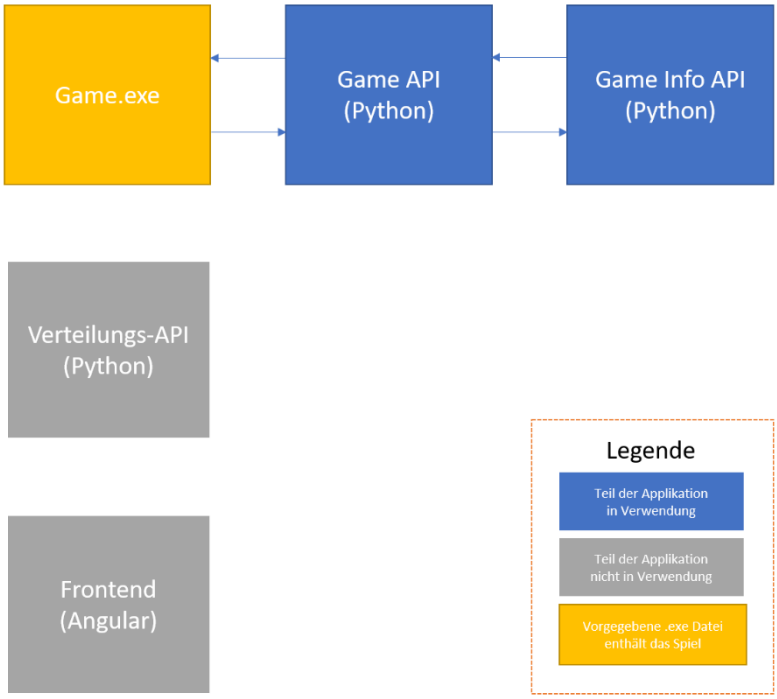

Abbildung 8 Genutzte Teilapplikationen und deren Zusammenhang im automatischen Spielmodus

Im automatischen Spielmodus (siehe Abbildung 8) schickt die game.exe ihre Aufrufe an die zentrale Game-API. Diese ermittelt auf Basis der internen Algorithmen und der Datenaufbereitung der Game-Info-API eine Aktion und schickt diese an die game.exe zurück.

3.2.2 Manueller Spielmodus

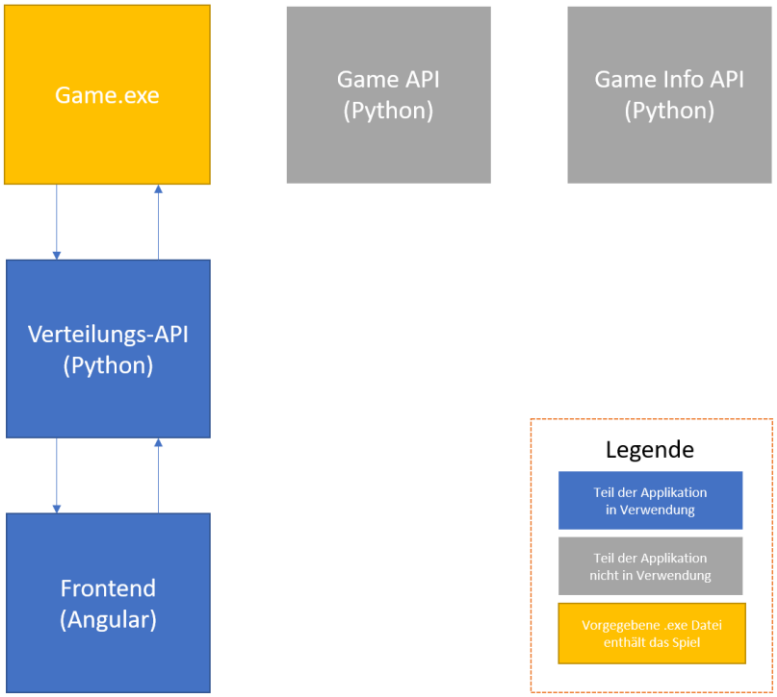

Abbildung 9 Genutzte Teilapplikationen und deren Zusammenhang im manuellen Spielmodus

Im manuellen Spielmodus (siehe Abbildung 9) werden die Aufrufe der game.exe an die Verteilungs-API, wo der aktuelle Spielstatus zwischengespeichert wird. Das Web-Frontend ruft anschließend die Verteilungs-API auf, um den Spielstatus zu laden und zu visualisieren. Nach der Aktionswahl des Nutzers im Web-Frontend wird diese über die Vermittlungs-API an die game.exe weitergegeben und somit ausgeführt.

3.2.3 Automatischer Spielmodus mit Visualisierung

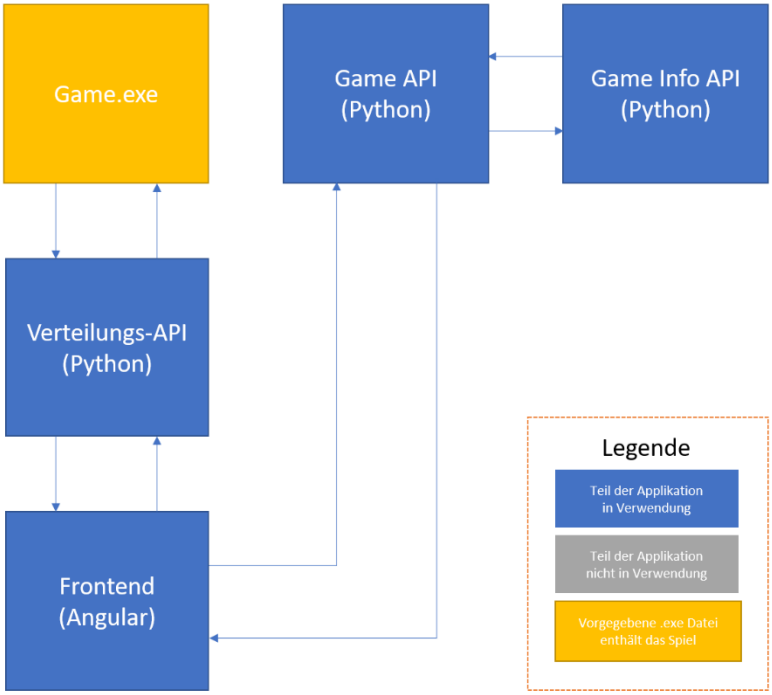

Abbildung 10 Genutzte Teilapplikationen und deren Zusammenhang im automatischen Spielmodus mit Visualisierung

Im automatischen Spielmodus mit Visualisierung (siehe Abbildung 10) gleicht der Verlauf dem des manuellen Spielmodus. Lediglich die Aktionsauswahl findet hier nicht durch den Nutzer, sondern durch einen Aufruf an die Game-API statt.

4. Game-API

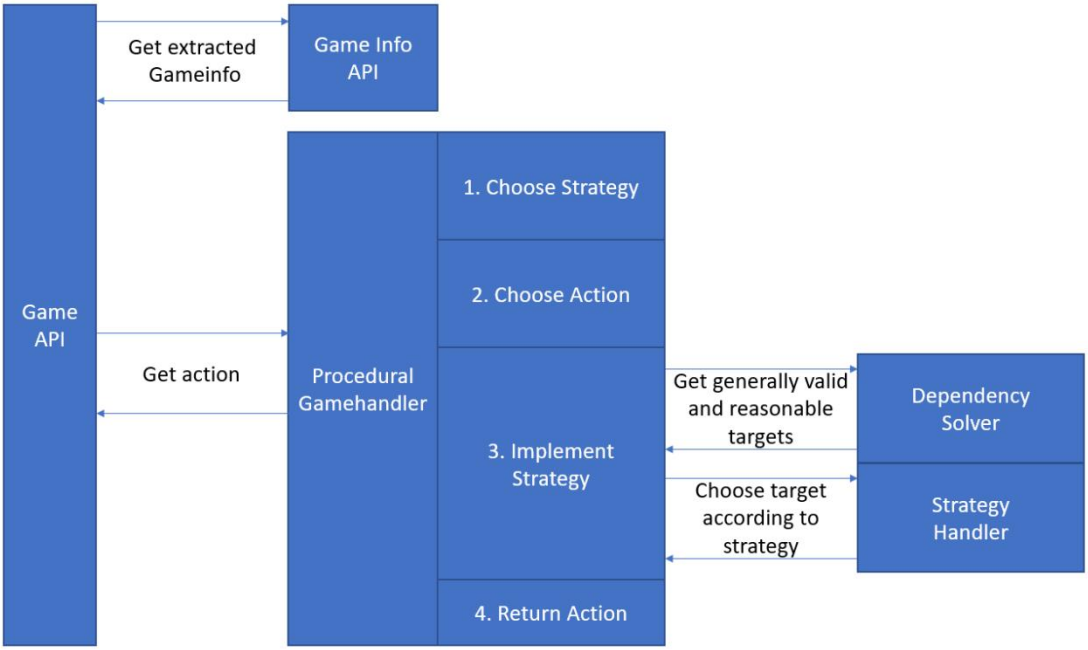

Abbildung 11 Aufbau Game-API

Die Game-API beinhaltet die Algorithmen zum automatischen Lösen des Spiels. Abbildung 1 zeigt den Ablauf des Lösungsprozesses von oben nach unten und von links nach rechts. Zunächst wird die Game-Info-API aufgerufen, um extrahierte Informationen über das Spiel zu erhalten. Anschließend wird der *Procedural Gamehandler* aufgerufen.

4.1 Technologie

Die Game-API ist als Web-API in der Programmiersprache Python realisiert. Dieser Aufbau erlaubt das Erfüllen der Anforderungen an das Programm im Hinblick auf die Zustandslosigkeit.

4.2 Ablauf

1. Schritt: Strategieauswahl

Der *Procedural Gamehandler* wählt im ersten Schritt anhand der extrahierten Informationen eine sinnvolle Strategie für die aktuelle Runde.

2. Schritt: Aktionsauswahl

Anhand der Strategie und den gegebenen Spielinformation wird eine Aktion gewählt. Dabei kann eine Aktion eindeutig durch die Strategie gewählt werden oder es wird zwischen Aktionen abgewägt (siehe mehr unter Foreseer unten).

3. Schritt: Strategieimplementierung

Im nächsten Schritt findet die Implementierung der Strategie statt. Dazu dienen ein allgemeines *Dependency Solver* Objekt und strategiebezogene *Strategy Handler* Objekte. Der *Dependency Solver* hilft dabei, grundlegende Auswahlmöglichkeiten für die Ziele der Aktion zu bestimmen. So muss zum Beispiel für die Quarantäne eine Stadt ausgewählt werden, hierfür liefert der *Dependency Solver* eine Liste von Städten. Hierbei werden auch allgemeine Filterungen angewendet. Wenn zum Beispiel die Aktion *deployVaccine* gewählt wurde, so filtert der *Dependency Solver* für die Auswahl der Städte diejenigen Städte, die für das entsprechende Pathogen bereits geimpft sind. Aufbauend auf diese allgemeine Auswahl bewertet ein *Strategy Handler* mithilfe von Gewichtungen oder Auswahlen, die die Strategie bedingen.

4. Schritt: Aktionsrückgabe

Durch die vorangegangenen Schritte wurde eine valide und der Strategie entsprechenden Aktion gewählt, die nun an die game.exe zurückgegeben und so ausgeführt werden kann.

Foreseer Objekt

Für die Verbesserung der Aktionswahl im zweiten Schritt kann ein Foreseer Objekt verwendet werden. Dieses durchläuft für verschiedene Aktionen, die verglichen werden sollen, den Strategieimplementierungsschritt, um die Ziele zu bestimmen, die nach der Strategie gewählt werden würden. Daraufhin berechnet es eine Wertung der Aktionen und gewichtet diese, um zu bestimmen, welche Aktion tatsächlich gewählt wird. So wird zum Beispiel zwischen *deployMedication* und *deployVaccine* abgewägt, indem ausgerechnet wird, wie viele Menschen aktuell durch ein Medikament in der nach der Strategie best gewählten Stadt effektiv circa gerettet werden und wie viele sich durch eine Impfung voraussichtlich retten ließen.

4.3 Strategien

Im Folgenden werden die Vorgehensweisen der wichtigsten unterschiedlichen Strategien sowie die Voraussetzungen für die Wahl der entsprechenden Strategie skizziert. Wichtig ist zu erkennen, dass die Game-API zustandslos ist. Das bedeutet, dass eine Strategie nicht zu Spielbeginn gewählt und dann das ganze Spiel über angewandt wird. Stattdessen wird bei jedem Aufruf der game.exe, also nach jeder ausgeführten Aktion, der aktuelle Spielstand neu bewertet und dementsprechend eine Strategie gewählt.

4.3.1 Strategie *Save Tokio*

Die grundlegende Idee dieser Strategie ist es, die bevölkerungsreichste Stadt abzuschotten und vollständig vor einem oder mehreren Pathogenen zu bewahren. Dazu Spielbeginn und meistens auch im Spielverlauf Tokio die bevölkerungsreichste Stadt ist, ist die Strategie nach Tokio benannt.

Voraussetzungen

Voraussetzung dieser Strategie ist das Vorhandensein von einem oder mehreren Pathogenen mit sehr hoher *Lethality* (++) und genug *Mobility* (üblicherweise > -), sowie einer niedrigen *Duration* (< o). Unter diesen Bedingungen lässt sich die Ausbreitung von zumindest einem der Pathogenen meist nicht verhindern. Auch eine Entwicklung von Impfung oder Medikamenten kann durch die niedrige *Duration* nicht rechtzeitig bewerkstelligt werden.

Vorgehen

Unter diesen Bedingungen bleibt nur eine Möglichkeit, das Spiel nicht zu verlieren. Da sich das Pathogen nicht einschränken oder bekämpfen lässt, bleibt nur die Möglichkeit, durch Quarantäne möglichst viele Menschen zu retten. Zu Spielbeginn hat man nur genug Punkte dafür, eine Stadt, also Tokio, dauerhaft unter Quarantäne zu setzen und das Pathogen in allen anderen Städten wüten und anschließend aussterben zu lassen. Im Normalfall überleben dadurch bei Anwendung der Strategie zu Spielbeginn von anfänglichen 100% circa 51% der Weltbevölkerung.

4.3.2 Strategie Shutdown Pathogen

Die Idee dieser Strategie ist es, ein bestimmtes Pathogen zu Beginn des Spiels einzudämmen, sodass sich alle anderen über die Karte verbreiten. Die Strategie macht sich dabei zunutze, dass eine Stadt nur von einem Pathogen gleichzeitig befallen sein kann.

Voraussetzungen

Diese Strategie ergibt nur Sinn, wenn es mehr als ein Pathogen gibt. Außerdem ist es notwendig, dass mindestens ein Pathogen eine ausreichende *Mobility* (> -) besitzt, um sich innerhalb einer oder zwei Runden in alle Städte zu verbreiten. Darüber hinaus müssen genug Punkte vorhanden sein, um alle Städte (im Normalfall nur eine oder zwei) des zu einschränkenden Pathogens über ausreichend Zeit (im Normalfall nur ein oder zwei Runden) unter Quarantäne zu setzen.

Vorgehen

Ziel der Strategie ist es, aus den Pathogenen so zu wählen, dass ein möglichst tödliches Pathogen zunächst eingeschränkt und dann ausgelöscht wird, bevor es sich verbreiten kann. Gleichzeitig sollte sich ein oder mehrere andere Pathogene in alle anderen Städte verbreiten. Das Pathogen *PV*, das sich möglichst verbreiten soll, wird also so gewählt, dass es die höchste *Mobility* und bei gleicher *Mobility* die geringste *Lethality* hat. Das Pathogen *PL* wird so gewählt, dass es die höchstmögliche *Lethality*, *Mobility*, *Duration* und *Infectivity*, in dieser Reihenfolge geordnet, hat. Die Städte, in denen PL vorkommt, werden nun unter Quarantäne gesetzt, bis keine Städte mehr ohne Pathogen sind. Anschließend wird für PL ein Medikament entwickelt. Da ein Pathogen in einer Stadt ausgelöscht wird, sobald es durch Medikamente unter 10% *Prevalence* fällt, werden bis zu dieser Grenze Medikamente auf alle Städte, die von PL befallen sind, angewendet, sobald das Medikament entwickelt wurde. Dieser Schritt wird allerdings erst dann angewendet, wenn es Städte gibt, die von keinem Pathogen befallen sind. So lassen sich Punkte sparen, sollte das Pathogen von allein schnell genug verschwinden. Eine Ausnahme hierfür gilt dann, wenn durch PL die Weltbevölkerung unter 50% zu fallen droht.

4.3.3 Strategie Vaccination and Medication

Die Idee dieser Strategie ist es, nacheinander für die tödlichsten Pathogene Impfung und Medikamente zu entwickeln. Anschließend wird versucht, die entsprechend optimalen Ziele zum Verteilen zu bestimmen.

Voraussetzungen

Diese Strategie ist sehr allgemein anwendbar und stellt eine Art Fallback-Strategie dar, die immer dann angewendet wird, wenn keine der anderen Sinn ergibt.

Vorgehen

Von den Pathogenen, die aktuell im Spiel auftreten, werden die tödlichsten ausgewählt, um für diese Impfung und Medikamente zu entwickeln. Zunächst wird dies für alle Pathogene mit einer *Lethality* von über o ausgeführt. Sind solche nicht oder nicht mehr vorhanden, werden im nächsten Schritt auch für Pathogene mit einer *Lethality* von - und -- gewählt. Nach Ablauf der Entwicklung werden gezielt in den bevölkerungsreichsten Städten Impfungen beziehungsweise Medikamente verteilt. Dabei wird über das oben bereits beschriebene Foreseer Objekt abgewogen, welche der beiden Aktionen voraussichtlich mehr Menschen rettet. Um diesen Schritt weiter zu optimieren, wird für die Städte anhand der Startrunde und der *Duration* des Pathogens berechnet, wann das Pathogen die höchste Prevalence erreicht, bevor es beginnt, Menschen zu töten. Medikamente werden dann bevorzugt zu diesem Höhepunkt verteilt, um die Gesamtanzahl an geheilten Menschen zu maximieren. Daraus ergibt sich aufgrund der prozentualen Heilung durch Medikamente ein erheblicher Unterschied.

4.3.4 Allgemeines Vorgehen

Einige Vorgehensweisen werden über alle Strategien hinweg angewendet.

Zurückhalten von Punkten

In jeder Strategie wird versucht, circa 50-80 Punkte zurückzuhalten, solang dies ohne die grundlegende Lahmlegung der Strategie möglich ist. Diese Anzahl an Punkten ist ausreichend, um eine Stadt, in der ein neues Pathogen auftritt, sofort unter Quarantäne zu setzen und eine Ausbreitung so sofort zu verhindern. Ähnlich der Strategie *Shutdown Pathogen* kann das Pathogen so sofort beseitigt oder ausgeharrt werden.

4.3.5 Zusammenfassung

Tabelle 2 Vergleich Lösungs-Strategien

Strategie	**Einsatz**	**Vorteil**	**Nachteil**
Save Tokio	Wenn ein oder mehrere Pathogene mit hoher *Lethality* und *Mobility* präsent sind	Kann es schaffen ein sonst unmöglich zu gewinnendes Spiel gerade so zu gewinnen	Kann nur selten angewendet werden
Shutdown Pathogen	Ersten Runde	Eines der Pathogene kann direkt geheilt werden	Wenn alle Pathogene ähnliche oder niedrige *Mobility* haben, kann Strategie fehlschlagen
Vaccination and Medication	Immer möglich	Hilft den allgemeinen Zustand der Infektion der Welt zu bessern	Kann nicht schnell genug auf Sonderfälle reagieren

4.3.6 Prioritäten der Strategien

Sollte die Strategiewahl mehrere Strategien für sinnvoll erachtet, wird folgende Priorisierung angewandt:

Save Tokio: Höchste Priorität, da diese Strategie, falls in der ersten Runde ein Pathogen mit hoher *Lethality* und *Mobility* präsent ist, die einzige Möglichkeit darstellt, das Spiel nicht zu verlieren.

Shutdown Pathogen: Zweithöchste Priorität, da mit dieser Strategie direkt zu Beginn des Spiels eines der Pathogene geheilt werden kann und somit im weiteren Verlauf des Spiels besser auf die weiteren Pathogene eingegangen werden kann.

Vaccination and Medication: Niedrigste Priorität, da diese Strategie allgemeine Situationen gut bewältigen kann.

5. Visualisierung

Die Grundlagen der interaktiven Systeme und der menschlichen Biologie und Psychologie lehren, dass die Augen das am besten entwickelte Sinnesorgan des Menschen sind und die beste Verknüpfung zum Gehirn haben [6]. Für das menschliche Verständnis einer Applikation ist eine gute Visualisierung daher maßgeblich.

5.1 Motivation

Der grundlegende Aufbau der Applikation für das automatische Lösen des Spiels besteht aus der game.exe, die in einem Terminal ausgeführt wird, sowie einer REST API. Eine API stellt dabei grundsätzlich eine Schnittstelle dar und sieht keinerlei Visualisierung vor. Das Terminal ist zwar auch eine Form der Visualisierung, allerdings nur in Textform. Gerade die komplexen Sachverhalte eines Spiels lassen sich in einer hunderte Zeilen großen JSON nicht überblicken. Für ein gutes Verständnis durch den Menschen ist eine gute Visualisierung allerdings nahezu unumgänglich. Eine Visualisierung hilft in diesem Fall in verschiedenen Hinsichten. Zum einen kann das Spiel manuell getestet werden, um bestimmte Verhaltensweisen zu erkennen und Auswirkungen von bestimmten Aktionen besser zu verstehen. Zum anderen hilft sie beim Debuggen und Testen der automatisierten Algorithmen.

5.2 Technologie

Technologisch ist die Visualisierung als Website auf Basis von Angular realisiert. Angular erlaubt es, sogenannte Single Page Applications zu implementieren. Dies bedeutet, dass die Applikation in verschiedene Komponenten aufgeteilt wird. Diese Komponenten werden nach Bedarf dynamisch geladen. Dadurch kann die Performance einer Applikation deutlich verbessert werden, da häufig nur kleine Teile neu geladen werden müssen, anstatt der gesamten Website. So bleibt zum Beispiel die Aktionsauswahl-Komponente dauerhaft gleich und muss nur einmal geladen werden.

5.3 Bestandteile

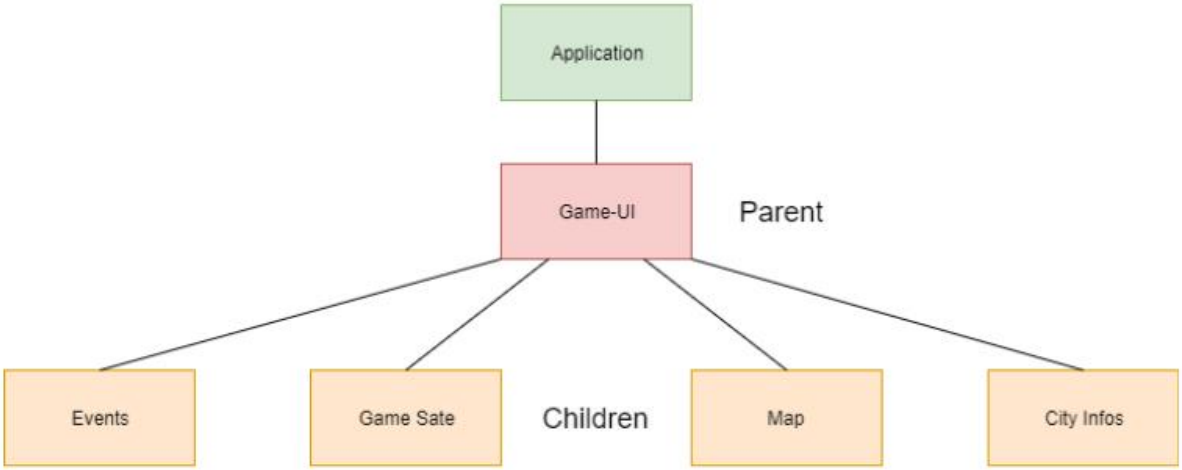

Abbildung 12 Bestandteile der Applikation

Die Applikation ist in verschiedene Bestandteile, technologisch gesehen verschiedene Komponenten, aufgeteilt. Es gibt fünf komplexe Komponenten, wovon eine Komponente als Parent dient (siehe Abbildung 12 Bestandteile der Applikation).

5.4 Game-UI

Die folgende Abbildung zeigt das gesamte User Interface mit allen Teilkomponenten. Die Teilkomponenten sind jeweils mit einem farbigen Rahmen markiert.

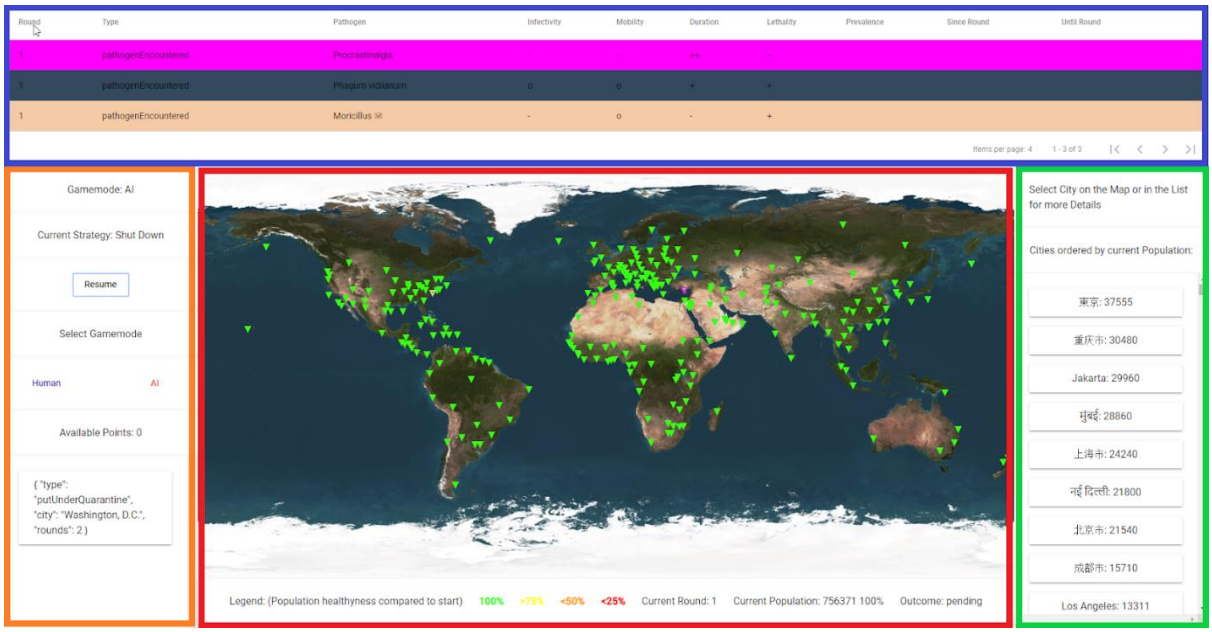

Abbildung 13 Aufbau Game-UI

5.4.1 Karte

Zentraler Bestandteil der Visualisierung ist die Karte (in Abbildung 13 Aufbau Game-UI rot umrandet). Sie spiegelt die Weltkarte wider, auf der alle im Spiel vorhandenen Städte als Punkte eingezeichnet sind. Für die Implementierung wurde hier die JavaScript Library Highcharts verwendet. Highcharts ist eine Scalable Vector Graphics (SVG) basierte multi-Plattform Diagramm Bibliothek, welche das Entwickeln von interaktiven, responsiven Web-Anwendungen ermöglicht.

Die Karte wird nach jeder Aktion aktualisiert. Falls Städte von Viren befallen sind, wird die jeweilige Stadt mit der Farbe des Virus gekennzeichnet, um die Ausbreitung der Viren übersichtlich betrachten zu können. Für das bessere Verständnis des Spiels werden alle Daten der jeweiligen Stände in die Karte geladen und können vom Benutzer durch Auswählen des Datenpunktes einer Stadt abgerufen werden. Durch die Selektion werden alle mit der ausgewählten Stadt verbundenen Städte hervorgehoben und mit Verbindungslinien verbunden. Dadurch kann der Benutzer einfacher die Zusammenhänge zwischen den Städten und eventuelle Ausbreitungen einsehen.

5.4.2 Events

Globale Events beinhalten Informationen über die aktuellen Viren, außerordentliche Vorkommnisse und über aktuelle Entwicklungen von Medizin und Impfstoff. Für eine einfachere Unterscheidung besitzt jeder Virus eine spezifische Farbe, welche auch so in der Karte visualisiert ist. Die globalen Events werden in einer Tabellenform visualisiert (in Abbildung 13 Aufbau Game-UI blau markiert). Durch Auswählen einer Zeile eines Virus, können alle Städte, welche von diesem Virus befallen sind, auf der Karte hervorgehoben werden.

5.4.3 Spielstatus- und Aktionskomponente

Die Spielstatus- und Aktionskomponente (in Abbildung 13 Aufbau Game-UI orange umrandet) ist die zentrale Schaltstelle, welche alle Aktionen, sei es die Kommunikation mit den APIs, die Verarbeitung der Benutzereingaben, die Anzeige der vorhergegangenen Aktionen oder das Management des Spielmodus, verwaltet. Ebenso wird natürlich auch von hier das Spiel gestartet und pausiert. Für eine bessere Bedienbarkeit des manuellen Spielmodus werden selektierte Städte automatisch als Aktionsselektion gespeichert. Außerdem hilft eine Autovervollständigung dem Benutzer dabei, eine

5.4.4 Stadtinfo-Komponente

Alle Infos einer Stadt werden in der Stadtinfo-Komponente (in Abbildung 13 Aufbau Game-UI grün umrandet) angezeigt. Diese können durch Selektieren einer Stadt in der Karte angezeigt werden. Dadurch werden sowohl die Eigenschaften der Stadt mit der aktuellen Bevölkerung als auch die Verbindungen sowie Städte Events dargestellt. Des Weiteren werden alle Städte nach aktueller Bevölkerung sortiert angezeigt, wodurch es einfacher ist, wichtige Städte schneller zu finden.

5.4.5 Theoretische Grundlagen des User Interface Designs

Usability Engineering

Tabelle 3 Usability Engineering vs Software Engineering

Usability Engineering	Software Engineering
Iterative Entwicklung	Klare, getrennte Phasen
Zyklen aus Analyse, Spezifikation, Prototyping und Evaluierung	Exakte Analyse und Spezifikation vor Beginn der Implementierung
Schrittweise Verfeinerung	DRIFT

Die Implementierung des User Interface (UI) basiert auf mehreren Grundlagen, welche ausschlaggebend für die Darstellung der Komponenten sind. Das Vorgehen unterscheidet sich bereits direkt zu Beginn der Entwicklung zum Vorgehen des klassischen Software Engineering. Es wird deshalb auch vom sogenannten Usability

Engineering gesprochen. Die wichtigsten Unterschiede zum Software Engineering sind die Entwicklungsphasen, die Analyse und Spezifikation und das Vorgehen, wie z.B. DRIFT (Do it right the first time) im Gegensatz zur Schrittweisen Verfeinerung [5]. Die folgende Tabelle (Tabelle 3 Usability Engineering vs Software Engineering) zeigt eine Gegenüberstellung der beiden Ansätze.

Visuelle Wahrnehmung

Bei der Visuellen Wahrnehmung wird zwischen der präattentiven und attentiven Wahrnehmung unterschieden. Die präattentive Wahrnehmung findet unterschwellig statt und sorgt dafür, dass spezielle Merkmale unmittelbar und schnell erkannt werden (<0,25 Sekunden) [6]. Darunter fallen zum Beispiel Objekte, die sich in Farbe oder Form signifikant von allen anderen dargestellten Objekten unterscheiden und sich somit deutlich abheben. Dies wird in der gegebenen Applikation zum Beispiel in Form der Symbole auf der Karte, der Buttons für die Spielmodi, die Farbdarstellung für die verbleibende Bevölkerung in den Städten und insbesondere die verschiedenfarbig dargestellten Viren genutzt.

Die attentive Wahrnehmungsphase folgt auf die präattentive Phase und beinhaltet die bewusste Suche nach bestimmten Elementen. Die dafür benötigte Dauer steigt linear mit der Anzahl der Elemente [6]. Es ist somit zu empfehlen, das UI nicht mit unnötigen Informationen zu füllen, sondern sich auf das Wesentliche zu beschränken. Dies wurde im UI der Applikation durch die verschiedenen Komponenten und die Interaktion mit diesen umgesetzt, wodurch komplexere Anzeigen erst durch die Interaktion des Nutzers sichtbar gemacht werden, um diesen auf den ersten Blick nicht zu überfordern.

Farbwahrnehmung

Das Auge des Menschen besitzt 10-mal so viele Rot- und Grünzapfen wie Blauzapfen, die für die Farbwahrnehmung zuständig sind, wodurch rote und grüne Farbtöne deutlicher wahrgenommen werden als blaue [7]. Dementsprechend ist es sinnvoll, für wichtige Elemente rote und grüne Farben einzusetzen. Dies wurde vor allem bei essentiellen Anzeigen wie der Anzeige der verbleibenden Bevölkerung in den Städten (Farbverlauf von Grün zu Rot), den Farben für die Pathogene und den wichtigen Knöpfen zur Auswahl des Spielmodus umgesetzt.

Gestaltgesetze

Um sich bestimmte Eigenschaften der visuellen Wahrnehmung des Menschen zunutze zu machen, werden allgemeine Gestaltgesetze umgesetzt. Hierbei geht es darum, Fragen wie “Wann wird Information als zusammengehörig empfunden?” oder “Wie wird die korrekte und schnelle Wahrnehmung von Informationen begünstigt” zu beantworten. Die wichtigsten Gesetze sind das Gesetz der Nähe und das Gesetz der Gleichheit [3].

Das Gesetz der Nähe besagt: “Räumliche Nähe führt dazu, dass Informationen als zusammengehörig wahrgenommen wird, selbst wenn sich Formen und Farben unterscheiden” [3]. Hierbei wird versucht, Unterschiede in der Bedienung durch Distanz und Zusammengehörigkeit durch Gruppierung zu vermitteln. In dem UI wurde deshalb eine klare Unterteilung der Komponenten vorgenommen, wie auch in der Tabelle 1 Spieleraktionen zum Eindämmen eines Pathogens zu sehen ist. Die Komponenten sind klar getrennt, wodurch der Aufgabenbereich der Komponenten klar verständlich sind.

Das Gesetz der Gleichheit trägt ebenfalls, aber in geringerem Maße, zur Wahrnehmung von Zusammengehörigkeit bei. Hierbei ist es wichtig, einheitliche Formen, Symbole, etc. zu verwenden, damit der Nutzer eine Zusammengehörigkeit des Produkts erkennt. Dies erhöht auch des Wiedererkennungswert und erleichtert die Bedienung [3].

Darüber hinaus gibt es weitere weniger wirksame Gestaltgesetze, wie z.B. das Gesetz der Geschlossenheit, welches besagt, dass Linien, die eine Fläche umschließen, unter sonst gleichen Umständen als eine Einheit aufgefasst werden [3]. Dieses Gesetz wurde verstärkt eingesetzt, um einerseits einzelne Teile des UI zu trennen und andererseits Teile zusammenzufassen, wie z.B. die Runden Historie, welche die getätigte Aktion, Strategie und Runde in einer Fläche zusammenfasst.

Feedback

Um unnötigen Stress und Verwirrungen sowie Unklarheiten zu vermeiden, ist klares Feedback von Aktionen nötig. Hierbei ist auch wichtig wie schnell diese Rückmeldung erfolgt und wie gut der Benutzer diese Rückmeldung interpretieren und wahrnehmen kann. Aus diesem Grund wurde im UI bei fast jeder Aktion eine Rückmeldung eingebaut, sei es das Warten auf den aktuellen Spielstand, das Filtern der Städte mit dem ausgewählten Pathogen, das Absenden von Aktionen oder das Zeichnen von Verbindungslinien zwischen den Städten.

Direkte Manipulation

Die Aktivierung von Kommandos orientiert sich am physischen Zeigen und Bewegen. Benutzer selektieren Icons, graphische Repräsentationen, die ihre Daten, Anwendungen oder den Systemzustand darstellen. Das System liefert unmittelbar eine klar erkennbare Rückkopplung, wie z.B. das Hervorheben von Buttons, Icons, etc. Direkte Manipulationen bieten den Vorteil, dass das zugrundeliegende Modell vereinfacht dargestellt wird [4]. Beispielsweise liefert das Hovern über die Punkte der Karte den aktuellen Städtenamen oder das Aufklappen der Baumstrukturen in den Städten die Details für die Verbindungen und Events.

6. Software Qualität

Zur Sicherstellung des konsistenten Funktionierens und Warten einer Software ist es wichtig, verschiedene Aspekte zur Sicherstellung der Software Qualität zu betrachten.

6.1 Software Testing

Um die Qualität von Software sicherzustellen und unerwartete Fehler möglichst zu vermeiden sind Software-Tests unverzichtbar.

6.1.1 API-Tests

Zentraler Bestandteil zum automatischen Lösen des Spiels ist die Game-API. Um bei Änderungen fortlaufend testen zu können, ob die grundlegende Funktionsweise des Lösens nicht unbeabsichtigt geändert wurde, enthält die Applikation API-Tests. Diese helfen dabei festzustellen, ob eine API insbesondere im Hinblick auf Funktionalität erwartete Ergebnisse liefert. So werden in den Tests zum Beispiel für verschiedene Spielstände geprüft, ob durch die Lösungsalgorithmen die gewünschte Strategie und Aktion gewählt wird.

6.1.2 Manuelle Tests

Kleinere Veränderungen einer Strategie oder der Strategieauswahl können bei der Lösung des Spiels erhebliche Auswirkungen haben. Diese beim Design oder Programmieren zu überblicken ist aufgrund des maßgeblich veränderten Spielverlaufs unmöglich. Insbesondere beim Optimieren des Algorithmus auf bestimmte Spielsituationen ist manuelles Testen daher unumgänglich. Sie können aufgrund der bereits beschriebenen umfangreichen Visualisierung sehr einfach durchgeführt werden. Der Tester kann ein Spiel automatisch lösen und sich dabei den Spielverlauf anzeigen lassen. Dabei kann er das Spiel auch pausieren und manuelle Aktionen ausführen, um bereits beim Testen Ideen für ein verändertes Design des Algorithmus zu sammeln.

6.2 Coding Conventions

Für die Sicherstellung von einheitlichem und wartbarem Code, der sich an Standards der jeweiligen Programmiersprache orientiert, wurden bei der Entwicklung die Bibliotheken pylint für Python-Code und tslint für Typescript-Code verwendet. Diese warnen bei der Entwicklung vor Verstößen gegen allgemeine Standards der Programmiersprache und zwingen den Entwickler somit, sich an diese zu halten

6.3 Wartbarkeit

Durch das objektorientierte und granular aufgebaute Design der Applikationen lässt sich das System sehr leicht anpassen. Die Zuständigkeiten der Objekte der automatischen Lösung, unter Game-API bereits beschrieben, sind klar definiert und voneinander getrennt. Zum Hinzufügen einer neuen Strategie muss zum Beispiel lediglich eine neue Strategie-Klasse erstellt und diese im Procedural Gamehandler unter den gewünschten Bedingungen zur Anwendung aufgerufen werden. Abhängigkeiten von Aktionen wie die Wahl einer Stadt zur Anwendung, die in der neuen Strategie gewählt werden, müssen nicht explizit programmiert werden. Stattdessen wird über den ActionDependencySolver eine im Allgemeinen gute Wahl getroffen. Soll für die Strategie eine besondere Einschränkung oder Auswahl getroffen werden, muss diese lediglich im jeweiligen Handler der Aktion hinzugefügt werden. Für die Anpassung bereits bestehender Strategien muss analog nur in der jeweiligen Klasse verändert werden:

- Procedural GameHandler
    - Strategieauswahl
- Strategieklasse
    - Aktionsauswahl
- ActionDependencySolver
    - Allgemeine Abhängigkeitslösung
    - Nicht strategiebezogen
- Aktionshandler
    - Strategiebezogene Abhängigkeit Lösung

7. Bewertung

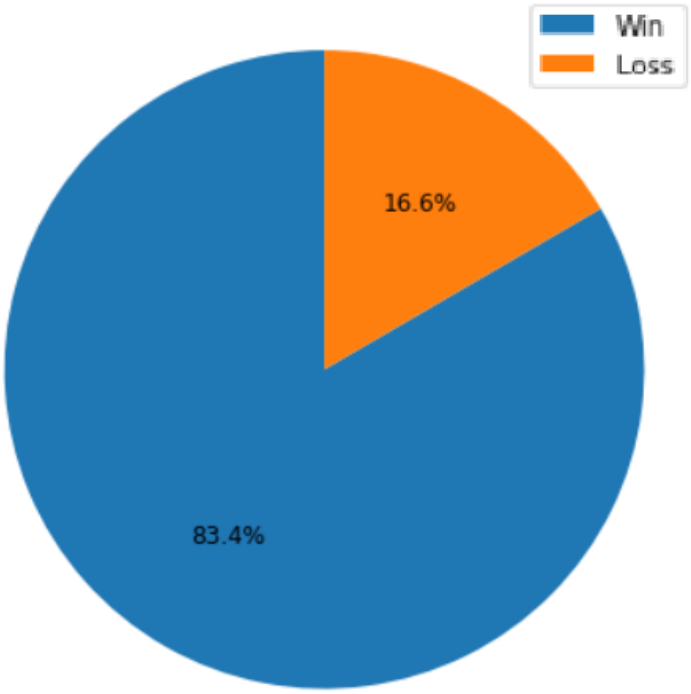

Abbildung 14 Sieg-Rate über 1000 Spiele

Die algorithmenbasierte Lösung erreicht bei der Auswertung von 1000 verschiedenen Spielen eine Sieg-Rate von 83,4% (siehe Abbildung 14 Sieg-Rate über 1000 Spiele). In Anbetracht dessen, dass es viele Spielsituationen gibt, die eine sehr spezifische Aktionsfolge voraussetzen und sich nicht durch eine generalisierte Strategie abdecken lassen, ist dies ein solides Ergebnis. Insbesondere ist allerdings hervorzuheben, dass die Lösung eine Visualisierung liefert, die einen Spielverlauf übersichtlich darstellt und es dem Nutzer so erlaubt, die ablaufenden Aktionen und deren Konsequenzen nachzuvollziehen. Die Lösung bietet somit eine gute Grundlage, die durch verschiedene Erweiterungen verbessert werden kann, wobei die Visualisierung stets Auskunft über die Folgen der Veränderung liefert.

8. Diskussion

Die gegebene Lösung versucht, für verschiedene Szenarien unterschiedliche Strategien anzuwenden und so eine möglichst große Abdeckung zu erreichen. Dadurch wird versucht, für möglichst viele Spielsituationen durch eine Aktionsfolge bereitzustellen, die zu einem guten Ergebnis führt. Da das Spiel eine Art Optimierungsproblem mit hoher Komplexität darstellt, kann dieser Ansatz keine optimale Lösung für jede Spielsituation finden. Es gibt jedoch viele Verbesserungsmöglichkeit.

8.1 Erhöhung der Zahl der Strategien

Der Ansatz von verschiedenen Strategien für verschiedene Spielsituationen lässt sich beliebig erweitern und granularer aufbauen. Durch die Anwendung von mehr sich signifikant unterscheidenden Strategien wird sich die Abdeckung bei sinnvoller Anwendung dieser natürlicherweise erhöhen und das Ergebnis verbessern. Betrachtet man allerdings die Menge an sich unterscheidenden möglichen Spielsituationen, so ist die Entwicklung von genug Strategien für eine vollständige Abdeckung unrealistisch.

8.2 Lokale Optimierung

Die Lösung ist an vielen Stellen abhängig Parametern, die unter einer bestimmten Annahme gewählt wurden. So ist zum Beispiel die Strategieauswahl auf Erfahrung und Auswertung von Spielsituationen basierend gesteuert. Viele dieser Parameter, und insbesondere das Zusammenspiel der mehrerer Parameter, lassen sich auf unterschiedliche Weise optimieren.

8.2.1 Einsatz von künstlicher Intelligenz

Auch wenn die Anwendung von künstlicher Intelligenz auf das Gesamtproblem nicht praktikabel ist, so könnte sie doch zur Lösung von Teilproblemen genutzt werden. Beispielsweise ist es sehr schwierig festzulegen, welche Strategie zu welcher Zeit angewendet werden sollte. Auf Basis von Massendaten, generiert durch das Anwenden jeder Strategie auf verschiedene Spielsituationen, könnte eine überwachte KI die Strategiewahl lernen. Hierbei ergibt sich zwar immer noch das Problem des Labelings, allerdings in einem deutlich weniger komplexen Umfang.

8.2.2 Einsatz von Evolutionären Algorithmen

Einige Parameter lassen sich nur näherungsweise berechnen und sind dabei vermutlich weit vom Optimum entfernt. Zum Beispiel wird bei der Abwägung zwischen dem Verteilen von Medikamenten und Impfungen berechnet, wie viele Menschen jeweils gerettet werden. Dabei lässt sich dies für Medikamente recht genau aus der *Prevalence* einer Stadt, der *Lethality* des Virus und anderen Variablen berechnen. Für die Impfung ist dies allerdings schwieriger, da eine Stadt möglicherweise nie vom entsprechenden Virus getroffen werden würde. Die korrekte Gewichtung dieser Parameter ließe sich durch Näherungsverfahren wie Evolutionäre Algorithmen optimieren.

8.2.3 Vorausberechnung der Verbreitung von Pathogenen

Ein großes Optimierungspotenzial liegt darin, zu erkennen, ob und wie sich ein Pathogen ausbreiten wird, und daraus Ableitungen über notwendige und verschwendete Aktionen zu treffen. Dadurch ließen sich unnötige Impfungen, Quarantäne, etc. vermeiden. Darüber hinaus kann eine Stadt nicht zweimal vom selben Pathogen befallen sein, wodurch sich weitere Sparmaßnahmen umsetzen lassen. Hierbei stellt sich allerdings die Frage, wie eine solche Berechnung aussehen könnte. So passiert es zum Beispiel häufig, dass sich zwei Pathogene jeweils in circa die Hälfte aller Städte verbreitet. Schafft man es nicht, die Pathogene auszulöschen, so passiert es häufig, dass eine Art Wechsel stattfindet und alle Städte, die zuvor von einem Pathogen befallen waren, nun vom anderen betroffen sind und umgekehrt. Wendet man eine Strategie an, die auf Impfungen basiert, ist jede Aktion zu diesem Zeitpunkt verschwendet. Allerdings lässt sich dieser Wechsel nicht ohne Weiteres zurückverfolgen, das eine Stadt keine Historie besitzt, aus der sich ablesen lässt, von welchen Pathogenen sie bereits befallen war

Quellenverzeichnis

[1] Michael Chui, James Manyika, Mehdi Miremadi. Januar 2020. *What AI can and can't do (yet) for your business*. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/what-ai-can-and-cant-do-yet-for-your-business

[2] Mauro Comi. Januar 2020*. How to teach AI to play Games: Deep Reinforcement Learning.* https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a

[3] Unbekannt Gestaltungsgesetze für Digitale Produkte. Januar 2020. https://userinterfacedesign.ch/gestaltgesetze/

Literaturverzeichnis

[4] T. Grossmann und R. Balakrishnan (2005) *A Probalistic Approach to modeling two-dimensional pointing*

[5] Christian Moser (2012) *User Experiance Deisgn Mit erlebniszentrierter Softwareentwicklug zu Produkten, die begeistern.* Springer

[6] Bellebaum C., Thoma P., Daum I. (2012) *Visuelle Wahrnehmung: Was, Wo und Wie. In: Neuropsychologie.* VS Verlag für Sozialwissenschaften

[7] Claudio Roller (2016) *Farbe und Repräsentation: Eine philosophische Studie zur Farbwahrnehmung.* Könighausen u. Neumann