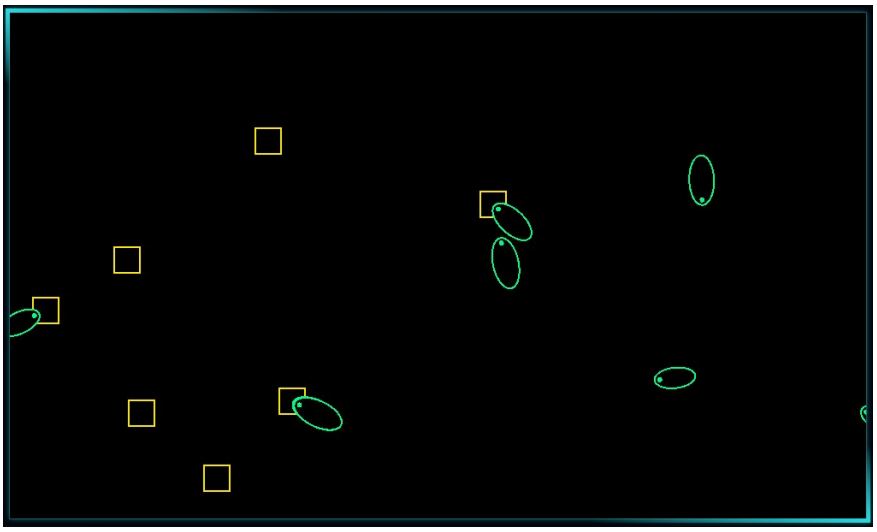
ALoAA

Artificial Life of Autonomous Agents

(Das künstliche Leben autonomer Agenten)

Projektdokumentation



Inhaltsangabe

Ausgangssituation

Für ein Berufsschulprojekt soll eine Simulation mit einer Animation implementiert werden, die zwei Zufallsparameter nutzt, eine Gleichverteilung und eine Normalverteilung. Zusätzlich muss der Benutzer mindestens 5 verschiedene Werte eingeben können und die Simulationsgeschwindigkeit in mindestens 3 Stufen regeln können.

Projektbeschreibung

Das Programm simuliert das Verhalten autonomer Agenten. Diese können sich durch Zellteilung vermehren, wobei einige ihrer Eigenschaften mutieren und dadurch Auswirkungen auf das Verhalten und den Ausgang der Simulation haben.

Projektziele

Die Simulation

Die Simulation besteht aus 2 Einheiten, im folgenden nur noch Entitäten genannt, den Agenten und Ressourcen. Es kann bis zu drei verschiedene Agenten geben, die durch die Ernährungsweise unterschieden werden, in autotroph, heterotroph und mixotroph. Außerdem gibt es eine Ressource, die die zwei Zustände organisch und anorganisch annehmen kann.
Alle Entitäten besitzen verschiedene Eigenschaften. Es gibt Eigenschaften mit festen Werten (Traits), die sich nur bei Mutationen ändern. Dann gibt es die veränderlichen Eigenschaften (Bars), bei denen der Wert zwischen einem festgelegten Minimum und Maximum schwanken kann. Schließlich gibt Fähigkeiten (Abilities), die die Entitäten ausführen können.
Die Entitäten befinden werden selber nur durch Grafik-Primitive wie Rechteck und Ellipse dargestellt. Ihre Umgebung wird hingegen überhaupt nicht visualisiert und dient nur zur Begrenzung ihrer Bewegung.

Die Benutzeroberfläche

Die Benutzeroberfläche besteht aus zwei Bereichen. Einer ist für die Konfiguration bestimmt, also zur Eingabe von Werten durch den Benutzer und der andere für die Anzeige der Simulation mit den Bedienelementen zur Steuerung der Geschwindigkeit.

Projektabgrenzung

Das Programm soll keine komplexe Simulation werden. Das heißt, dass nur wenige Eigenschaften für das Verhalten betrachtet werden, nämlich die Ernährungsform und Geschwindigkeit.
Es soll ein Kreislauf dargestellt werden. Es dürfen also außer den Starteinheiten dem System keinerlei weitere Agenten oder Ressourcen hinzugefügt werden.
Die Entities sollen nicht komplex dargestellt und animiert werden, unter anderem um mehr Prozessorkapazität und Arbeitsspeicher zur Verfügung zu haben, sodass das Programm auch auf leistungsschwachen Rechnern läuft oder leistungsstarke mehr Entities simulieren können.
Ebenso soll die Umgebung so simpel wie möglich dargestellt werden und auf zusätzliche Effekte wie Tageszeit oder Wetter verzichtet werden.
Die Intelligenz der Agenten soll nicht mittels maschinellem Lernen wie NEAT[1] erfolgen, sondern durch eine einfache Logik, wie in Abbildung 2 dargestellt ist, implementiert werden.

Projektplanung

Projektphasen

Für das Projekt sind 120 Stunden angesetzt in einem Zeitraum vom 06.09.2022 bis zum 13.01.2023, was 18 Wochen und 2 Tagen entspricht.
In Tabelle 1 ist ein grober Zeitplan dazu aufgeführt.

Phase	Geplant
Analyse	20
Entwurf	30
Implementation	60
Dokumentation	10
Gesamt	120

Tabelle 1

1 https://www.cs.ucf.edu/~kstanley/neat.html

Ressourcenplanung

Hardware

Für die Entwicklung wird natürlich ein Computer (Laptop oder Desktop-PC) benötigt, idealerweise zwei Bildschirme für ein angenehmeres und effizienteres Arbeiten, ein zusätzlicher Bereich zum Schreiben von Notizen und Entwurfsskizzen. Das ganze sollte in einem abgetrennten Bereich, noch besser in einem eigenen Raum für eine bessere Arbeitsatmosphäre stattfinden.

Software

Folgende Programme der Entwicklungsumgebung müssen darauf zur Verfügung stehen:

- Visual Studio Code mit den Erweiterungen
  - ESLint[2] und sonarlint[3] für eine automatisierte Clean-Code-Prüfung und
  - der ESLint Style Guide Airbnb für einen einheitlichen Codestil
  - Cypress[4] für die automatischen Unit Tests
  - webpack[5] zur Bündelung der JavaScript-Dateien in ein einziges Bundle
- Node.js[6] wird für Cypress und webpack benötigt
- PixiJS[7] für die Grafikausgabe der Animation
- Math-Extras Bibliothek von PixiJS für die Berechnungen mit Vektoren
- Chart.js[8] für die Darstellung der Diagramme
- Firefox und Chrome zum Testen der Software auf der Zielplattform

Entwicklungsprozess

Als Vorgehensmodell wurde eine Kombination aus den zwei agilen Modellen Scrum[9] und Extreme Programming[10] erstellt, die gezielt auf Contiuous Integration und Continuous Deployment (CI/CD) ausgelegt ist, das heißt, dass immer eine lauffähige Version des Programms zur Verfügung steht. Eine schematische Darstellung zum daraus resultierenden Arbeitsablauf befindet sich in Abbildung 1.

2 https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint
3 https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarlint-vscode
4 https://www.cypress.io/
5 https://webpack.js.org/
6 https://nodejs.org/en/
7 https://pixijs.com/
8 https://www.chartjs.org/
9 https://www.scrum.org/resources/what-is-scrum
10 http://www.extremeprogramming.org/

Durch die Scrum-Methode sollte sichergestellt werden, dass schnell und flexibel auf Veränderungen reagiert werden konnte. Die Regeln des Extreme Programming[11] sollten zu einem guten Software-Design und zu Clean Code führen.
Folgende Regeln wurden genutzt:

- verwende User Stories
- erstelle kleine Releases
- beginne jede Iteration mit der Iterationsplanung
- implementiere die einfachste Lösung
- füge eine Funktionalität erst hinzu, wenn sie gebraucht wird
- führe wenn immer möglich ein Refactoring durch
- erstelle die Unit Tests zuerst
- alle Unit Tests müssen vor einem Release erfolgreich bestanden sein

Analysephase

Ist-Analyse

Das Projekt soll als Web-Applikation umgesetzt werden, deshalb ist die am meisten verbreitete Web-Sprache JavaScript für diesen Einsatz ausgewählt worden. Für sie stehen online zahlreiche unterschiedliche Open Source Bibliotheken und Frameworks mit ihren Dokumentationen zur Verfügung.

In der Entwicklungsumgebung Visual Studio Code wurde bereits viel Erfahrung gesammelt, auch mit den bereits erwähnten Lintern und Node.js.

Neu war das Framework webpack zum Bündeln der Programmressourcen, aber durch die gute Online-Dokumentation und die Beschränkung auf die einfachsten Funktionen, wurden mögliche Probleme minimiert.
Ebenfalls neu war Chart.js für die Diagrammdarstellung. Aber auch hier gibt es eine gute API-Beschreibung und viele YouTube-Tutorials dazu. Damit und wieder durch eine Einschränkung der zu nutzenden Features wurden möglich Fehlerquellen reduziert.
Die Grafikbiblothek PixiJS und das Test-Framework Cypress wurden noch nicht sehr häufig eingesetzt, aber auch hier wurde sich nur auf bestimmte Features festgelegt, wodurch die Recherche zielgerichteter durchgeführt werden kann.

11 http://www.extremeprogramming.org/rules.html

Anforderungsanalyse

Funktional Anforderungen

Was das fertige Projekt leisten soll:
Das Programm soll die Evolution aller Entities berechnen und graphisch darstellen.
Der Benutzer soll durch Eingabeparameter die Simulation und damit das Ergebnis beeinflussen können.
Während der Simulation soll die Simulationszeit verlangsamt oder beschleunigt werden können.
Zusätzlich sollen während der Simulation sollen Ergebnisse statistisch aufbereitet in der Benutzeroberfläche mittels Diagrammen angezeigt werden.
Hierfür soll ein Diagramm die aktuelle Anzahl aller Entitäten als Zeitliniendiagramm ausgeben und ein weiterer die Masse, die durch die jeweilige Entitätsgruppe erzeugt wird als Balkendiagramm.
Deshalb soll alles in einem Fenster dargestellt werden, sodass die Simulationsdarstellung parallel zu den statistischen Erhebungen erfolgen kann.
Um den evolutionären Aspekt, also die Mutation, wirklichkeitsnäher zu realisieren, sollen die Zufälle einer Normalverteilung folgen, die

Nichtfunktionale Anforderungen

Das Programm sollte möglichst viele Clean Code Regeln[12] einhalten, wie *Keep It Simple and Stupid*, *Dont‘ Repeat Yourself, Single Responsibility Principle* oder *Separation of Concerns,* um nur ein paar aufzuzählen.
Komplexe Aufgaben wie die Animation oder die Darstellung der Diagramme sollte durch Verwendung von entsprechenden Bibliotheken so weit wie möglich vereinfacht werden.
Um einen stetigen Fortschritt zu unterstützen, sollte nach jedem Commit, entsprechend dem Vorgehensmodell eine lauffähige Version vorliegen.

Risikoanalyse

Für die Risikoanalyse wurden mögliche Risiken untersucht und tabellarisch in Tabelle 2 erfasst. Für jedes Risiko wurde die Wahrscheinlichkeit für das Eintreten dieses Ereignisses abgeschätzt, ihre Auswirkung auf den erfolgreichen Projektabschluss bewertet und entsprechende Maßnahmen zur Minimierung ermittelt und durchgeführt.

12 https://t2informatik.de/wissen-kompakt/clean-code/

Qualitätsanforderungen

Die verwendete Grafikbibliothek ebenso wie die Diagrammbibliothek sollen vor allem auf Performance ausgelegt sein.
Die Unit-Tests sollen den gesamten Code abdecken und alle Tests müssen erfolgreich abgeschlossen sein.
Der Zufallszahlengenerator muss vor allem schnell sein und eine gute Gleichverteilung liefern.

Entwurfsphase

Zielplattform

Dadurch, dass das eine Web-Anwendung erstellt werden soll, muss diese auf einem Web-Browser laufen. Das Programm wird in erster Linie für Chrome und Firefox programmiert, sollte aber auch auf jedem anderen modernen Web-Browser lauffähig sein. Dementsprechend werden die Web-Programmiersprachen JavaScript, HTML und CSS verwendet.
Besondere Anforderungen an die Hard- und Software bestehen insofern, als ein aktueller Web-Browser benötigt wird und für eine Simulation mehr Prozessorleistung und Arbeitsspeicher immer zu von Vorteil sind.
Das Programm läuft ausschließlich im Browser des Anwenders. Es wird also kein Internetzugang benötigt.

Architekturdesign

Eine Animation sowie die Aktualisierung der Diagramme erfordert eine Programmschleife, die hier als Game Loop Pattern[13] (Spielschleife) umgesetzt wurde.
Die Ausgabe und Darstellung der Programmdaten während der Simulation benötigt eine Datenstruktur zum Zwischenspeichern aller für die Anzeige ausgewählten Werte. Dadurch, dass die genutzte Diagrammbibliothek die Daten als einfaches Array erwartet, werden alle Agenten und Ressourcen getrennt von einander in zwei globalen Arrays gespeichert, die in einer Klasse gekapselt wurden.
Auf eine Datenbank wurde vollständig verzichtet, weil keine Anforderung vorliegt, dass eine persistente Datenhaltung der Simulationsergebnisse benötigt wird.

13 https://gameprogrammingpatterns.com/game-loop.html

Maßnahmen zur Qualitätssicherung

Das Vorgehensmodell sowie die Einhaltung von Clean-Code-Methoden tragen schon maßgeblich zur Qualitätssicherung bei. Außerdem unterstützen zwei sogenannte Linter dabei, automatisch den Quellcode auf Clean Code zu überprüfen. ESLint und sonarlint kamen hierbei zum Einsatz, weil JavaScript als Programmier-sprache eingesetzt wurde.
Zusätzlich wurden die Schnittstellen aller verwendeter Funktionen der Grafikbibliothek automatisch getestet.
Des Weiteren wurde ein Blackbox-Test durchgeführt, für den die User Stories eines Benutzers als Testfälle dienten.
Sämtlicher Code sollte durch erfolgreiche Unit Tests abgedeckt werden. Besonderer Wert wurde dabei auf die Basisklassen gelegt.
Zusätzlich wurden Performance-Tests unter verschiedenen Zufallszahlen-generatoren durchgeführt, um den besten und effizientesten auszuwählen.

Implementierungsphase

Um das gewählte Vorgehensmodell[14] durchzuführen, wurden für den Beginn die Anforderungen (Stakeholder Requirements und System Requirments) aus der Anforderungsanalyse verwendet. Diese wurden soweit möglich in Gruppen (Epics) eingeteilt und in User Stories umformuliert. Diese dienten dann als Ausgangslage (Items im Backlog) für jede Iteration (Sprint) des Programms.

Vor jedem Sprint wurde eine Planung durchgeführt, um zu evaluieren, welche Items in dieser Iteration umgesetzt werden können. Dazu wurde überprüft, ob die gewählten User Stories das INVEST[15] Prinzip erfüllen und gegebenenfalls angepasst sowie daraus Akzeptanzkriterien für die User Stories abgeleitet. Der Aufwand der gewählten Stories wurde nach der Planungs-Poker-Methode abgeschätzt, um den gesetzten zeitlichen Rahmen von maximal vier Stunden für die Implementierung aller ausgewählten Anforderungen nicht zu überschreiten.
Diese Auswahl (Sprint Log) wurde dann in der Sprint-Phase nach dem Prinzip des Test Driven Development (TDD) durchgeführt. Ein Sprint war beendet, sobald alle Akzeptanztests erfolgreich abgeschlossen wurden oder die Zeit abgelaufen war.
In beiden Fällen schloss sich dann die Release-Phase an, die im erfolgreichen Fall erst den Development-Branch mit dem Master-Branch zusammenführte (merge), um danach die Sprint-Phase auszuwerten (Review) und entsprechend anzupassen. Falls die Zeit nicht ausreichte, wurde kein Merge, sondern nur ein Review durchgeführt. Alle Erkenntnisse aus dem Review veränderten mal

14 siehe Tabelle 2
15 https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

mehr, mal weniger den Backlog, in dem neue Anforderungen hinzukommen, zu große aufgeteilt oder obsolete entfernt werden.
Der erfolgreiche Branch konnte dann veröffentlicht werden. Das Feedback, das durch die Verwendung der Software durch Tester gewonnen wurde, floss wieder in die Anforderungssammlung ein.
Um das Prinzip von CI/CD noch weiter zu forcieren, wurde nur ein Branch für die aktuelle Iteration des Programms verwendet und nur ein einziger für den Entwicklungsprozess.
Allgemein sollten zuerst die Basiselemente implementiert werden, wie eine rudimentäre grafische Benutzeroberfläche (GUI) zur Visualisierung der Simulation ohne weitere Steuerelemente. Danach wurden die Klassen der Entities, ihre Eigenschaften und die Umgebung, in der sie simuliert werden sollen, hinzugefügt. Als Nächstes wurde die Grafikbibliothek zur Darstellung der Animation eingebunden. Im Anschluss kamen die Steuerelemente der GUI hinzu. Im nächsten Schritt wurde die Bibliothek, zur Erzeugung der Graphen, eingesetzt. Zuletzt wurden fehlende Elemente, wie zum Beispiel die Start- und Konfigurationsseite, hinzugefügt.

Fazit

Soll-/Ist-Vergleich

Der vorgegebene Zeitplan konnte nicht eingehalten werden, wie Tabelle 4 zu entnehmen ist. Ein Grund dafür ist eine schlechte zeitliche Abschätzung der Implementierungsphase, die mit User Stories und TDD durchgeführt wurde. Beim Einsatz dieser Techniken fehlt es noch an Erfahrung, den zeitlichen Rahmen besser vorherzusagen.
Außerdem kam bei der Überführung des Prototyps zu Problemen, die etwa 20 Stunden an Recherchen und Umsetzung mit TDD unnötig verschlungen haben. Weitere 5 Stunden wurden bei der Recherche zum Unit-Test-Framework verloren, weil es auch da zu Problemen beim Einbinden von Bibliotheken in die Testumgebung kam.
Durch diese Verzögerungen konnten einige wünschenswerten Features nicht umgesetzt werden und einige visuell störende Bugs nicht behoben werden.

Phase	Geplant	Tatsächlich	Differenz
Analyse	20	14	6
Entwurf	30	62	-32
Implementation	60	126	-66
Dokumentation	10	16	-6
Gesamt	120	218	-98

Tabelle 4

Erkenntnisse

Durch mehr Übung im Umgang mit User Stories und TDD kann mehr Geschwindigkeit und letzendlich größerer Nutzen erzielt werden. Auch die Benutzung mit der Entwicklungsumgebung, speziell im Zusammenspiel mit dem Test-Framework verbessert sich so.

Ausblick

Viele interessante Ideen wurden während der Anforderungsanalyse und der Implementierungsphase gesammelt, die nicht in der vorliegenden Version enthalten sind. Diese können als Ausgangslage für ein Folgeprojekt genutzt werden.
Auf der Startseite kann die Anzahl der Eingabeparameter erweitert werden und eine Funktion zum Abspeichern und Laden der Konfiguration hinzugefügt werden.
Ebenso könnte eine laufende Simulation abgespeichert und zu einem späteren Zeitpunkt fortgesetzt werden.
Natürlich können mehr oder sogar alle Eigenschaften der Entitäten von Mutationen betroffen sein.
Der Benutzer sollte die Möglichkeit haben, auf alle Informationen der Entitäten zugreifen zu können.
Genauso verbessert eine zusätzliche Seite, die mehr und vor allem interaktive Diagramme bereitstellt, die statistischen Auswertungen und erhöht den Nutzen der Software.

Anhang

Abkürzungsverzeichnis

BDD	Behavior-driven development
CI/CD	Contiuous Integration und Continuous Deployment
GUI	Graphical User Interface
NEAT	NeuroEvolution of Augmented Topologies
TDD	Test Driven Development

Abbildungen

Abbildung 1 – Vorgehensmodell

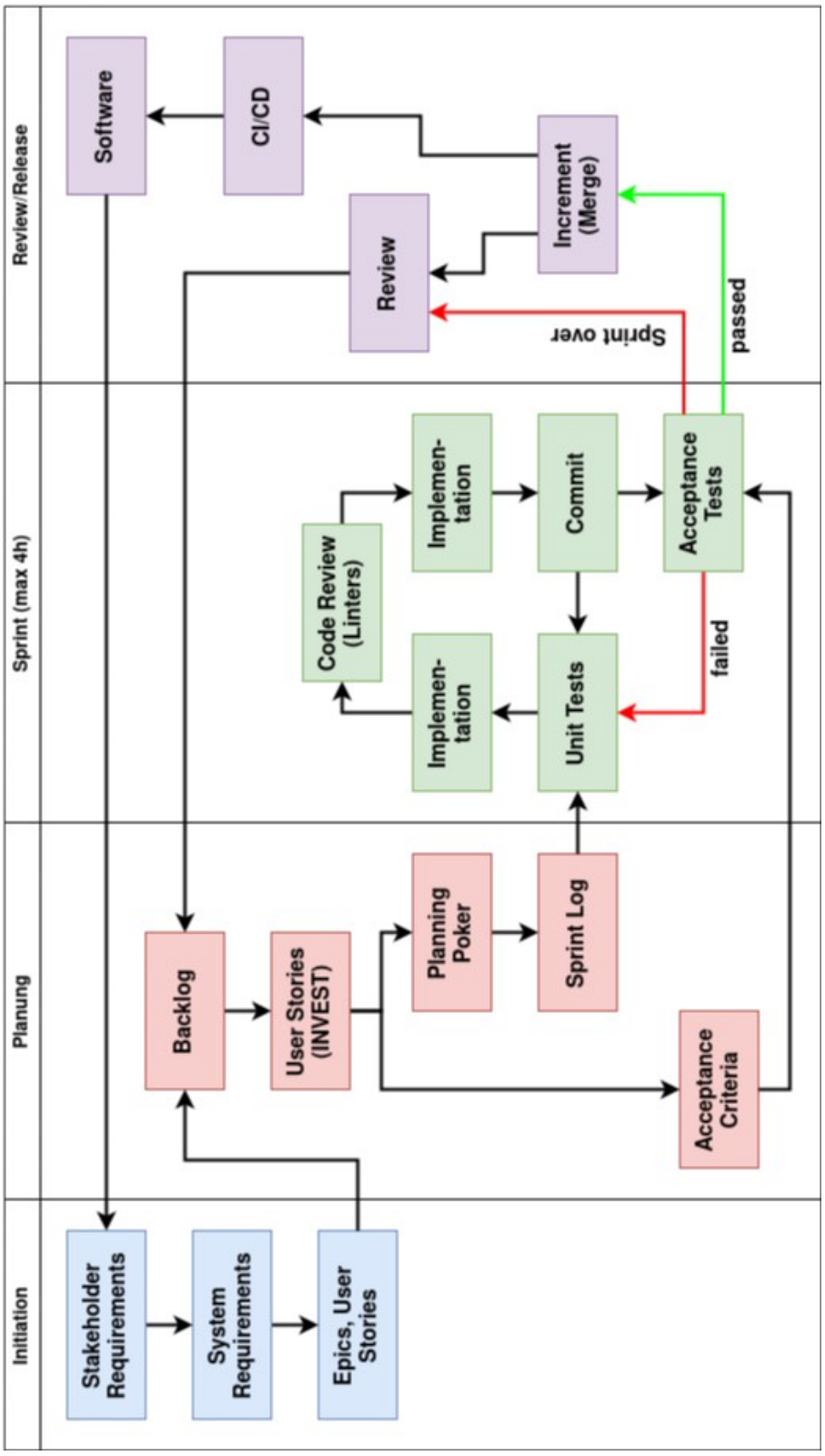
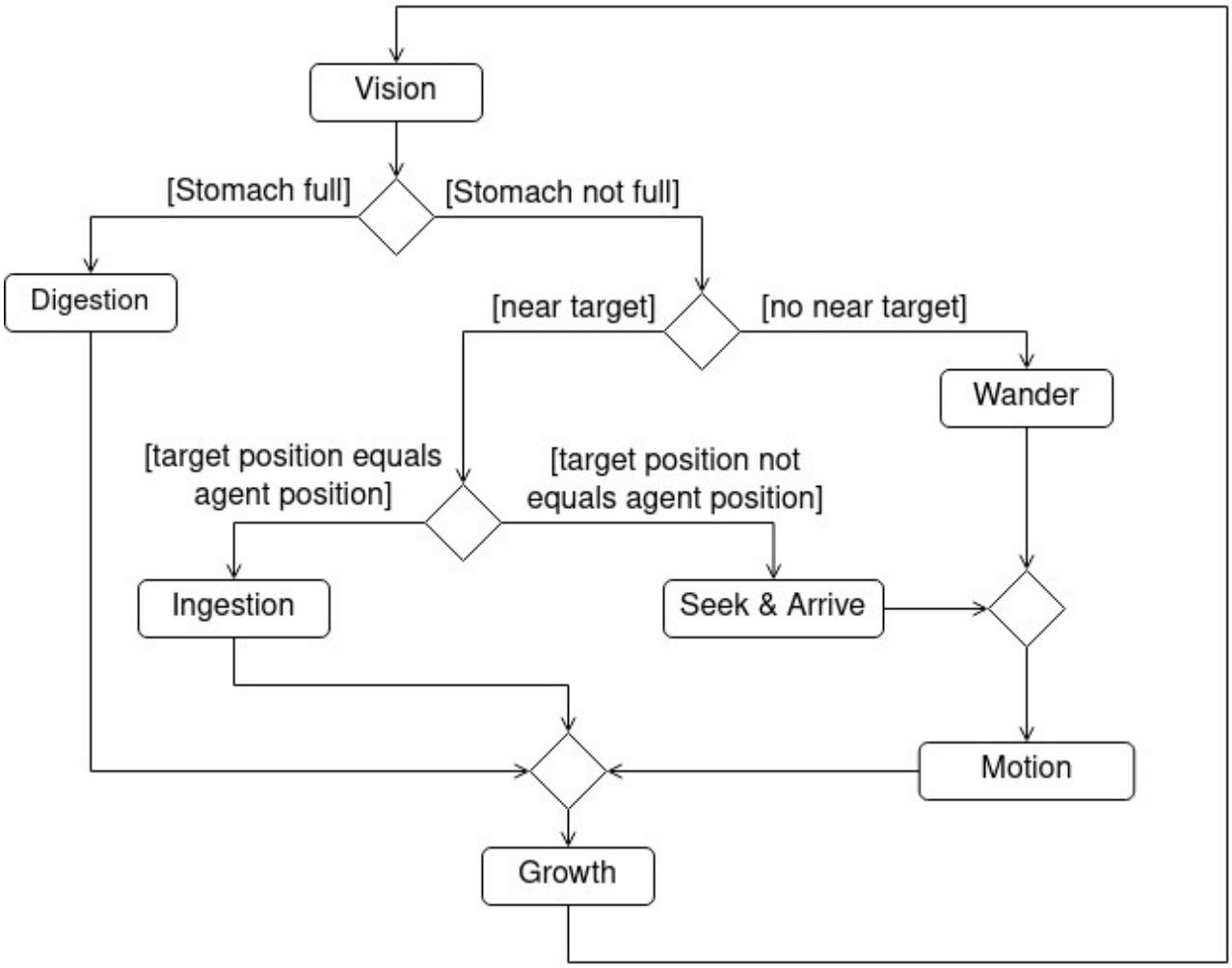


Abbildung 2 – Verhalten eines Agenten



Tabellen

Tabelle 2 – Risikoübersicht

ID	Risiko	Maßnahmen	Wahrschein-lichkeit	Bewer-tung[1]
1	Eigener Krankheitsausfall	Durch CI/CD immer eine lauffähige Version verfügbar, abgabebereit	gering	4
2	Krankheit von Partnerin/Kind	Durch CI/CD ist immer eine lauffähige Version verfügbar, abgabebereit	hoch	3
3	Verlust der Projektdaten	Zur Versionsverwaltung wird Github genutzt	gering	5
4	Zeitaufwendige Einbindung in Projekte des Ausbildungsbetriebs	Entsprechende Absprachen wurden mit dem Ausbilder getroffen	gering	3
5	Kein Zugriff auf Entwicklungsumge-bung durch Ausfall wichtiger Hardware wie Laptop	Ersatz Desktop-PC ist vorhanden und sämt-liche Einstellungen der Entwicklungsumgebung sind mit Github synchronisiert	gering	4
6	Projekt ist umfang-reicher als geplant	Durch CI/CD ist immer eine lauffähige Version verfügbar, abgabebereit	hoch	3
7	Methoden des Vorgehensmodells benötigen zu viel Zeit	Einige Methoden wie User Stories und TDD an passenderen Stellen auslassen	hoch	3

[1]siehe Tabelle 3

Tabelle 3 – Risikobewertung

1	zu vernächlässigen
2	unkritisch
3	In Bereichen kritisch
4	kritisch
5	projektgefährdend