MBA Strategic Management & Technology

TU WIEN ACADEMY FOR CONTINUING EDUCATION

# Zur Simulation von Effectuation Prozessen: Möglichkeiten und Grenzen einer agentenbasierten Computersimulation der Effectuation Theory illustriert am Beispiel des X-Pad-Projekts

Master Thesis zur Erlangung des akademischen Grades
“Master of Business Administration”

eingereicht bei
Prof. Dr. Jochen Koch

Rudolf Herold

00326824

Wien, 15.09.2023

# Eidesstattliche Erklärung

Ich, **RUDOLF HEROLD**, versichere hiermit

1. dass ich die vorliegende Master These, "ZUR SIMULATION VON EFFECTUATION PROZESSEN: MÖGLICHKEITEN UND GRENZEN EINER AGENTENBASIERTEN COMPUTERSIMULATION DER EFFECTUATION THEORY ILLUSTRIERT AM BEISPIEL DES X-PAD-PROJEKTS", 151 Seiten, gebunden, selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfen bedient habe, und
2. dass ich diese Master These bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Wien, 15.09.2023

_____
Unterschrift

## Kurzfassung

Im Kontext der Unternehmenssimulation bietet die Effectuation-Theorie wertvolle Einsichten in die Entscheidungsfindung von Unternehmenden. Diese Arbeit untersucht, inwieweit ein computergestütztes Simulationsmodell entwickelt werden kann, um alle Effectuation-Prozesse abzubilden, und welche Möglichkeiten und Grenzen sich bei der Implementierung dieses Modells ergeben. Mit einem agentenbasierten Simulationsmodell wurden die Kernprinzipien der Effectuation-Theorie abgebildet und evaluiert. Die Methodik umfasst sowohl die Modellierung der Agenten und ihrer Entscheidungsprozesse als auch die empirische Analyse der Simulationsergebnisse. Die Ergebnisse zeigen, dass das Modell in der Lage ist, verschiedene Szenarien und Entscheidungsstrategien erfolgreich zu simulieren. Allerdings sind die Resultate durch die begrenzte Datenbasis und die Abstraktionsebene des Modells eingeschränkt. Diese Arbeit leistet einen Beitrag zur theoretischen Fundierung der Effectuation im Rahmen der Unternehmenssimulation und bietet Einblicke in die praktische Anwendbarkeit von Simulationsmodellen. Sie zeigt jedoch auch die Limitationen aufgrund der Komplexität der realen Entscheidungsprozesse und der Abstraktion im Modell. Für zukünftige Forschungen wäre eine Erweiterung der Datengrundlage und eine Verfeinerung der Modellparameter wünschenswert, um die Generalisierbarkeit und Anwendbarkeit der Ergebnisse zu erhöhen. Diese Arbeit demonstriert die Machbarkeit und die inhärenten Beschränkungen bei der Modellierung von Effectuation-Prozessen und bildet eine Grundlage für weitere Forschung in diesem Bereich.

## Abstract

In the context of enterprise simulation, the Effectuation Theory offers valuable insights into entrepreneurial decision-making. This Master's thesis explores the extent to which a computer-based simulation model can be developed to capture all effectuation processes, and what opportunities and limitations arise in the implementation of this model. Using an agent-based simulation model, core principles of Effectuation Theory were modelled and evaluated. The methodology encompasses both the modelling of agents and their decision-making processes, as well as the empirical analysis of the simulation outcomes. The results demonstrate that the model is capable of successfully simulating various scenarios and decision-making strategies. However, the findings are constrained by the limited data foundation and the level of abstraction in the model. This work contributes to the theoretical underpinning of effectuation in the domain of enterprise simulation and provides insights into the practical applicability of simulation models. It also identifies limitations due to the complexity of real-world decision-making processes and the abstraction in the model. For future research, an expansion of the data foundation and refinement of the model parameters would be desirable to increase the generalizability and applicability of the findings. This thesis demonstrates the feasibility and inherent limitations of modelling effectuation processes and lays a foundation for further research in this area.

# Inhaltsverzeichnis

# 1 Einleitung

Die Unternehmenssimulation ist ein vielseitiges Werkzeug zur Modellierung, Analyse und Vorhersage von Geschäftsprozessen und -ergebnissen. In den letzten Jahren hat der Ansatz der Effectuation immer mehr an Bedeutung gewonnen, um unternehmerisches Handeln in unsicheren und dynamischen Märkten zu verstehen. Diese Masterarbeit untersucht die Schnittstelle zwischen Unternehmenssimulation und Effectuation, indem sie die Entwicklung eines computergestützten Simulationsmodells zur Abbildung von Effectuation-Prozessen erforscht.

## 1.1 Motivation

Die moderne Geschäftswelt ist geprägt von einer beispiellosen Dynamik und Unsicherheit, die Unternehmen vor komplexe Entscheidungsprobleme stellt. In diesem Kontext hat die Effectuation-Theorie, die erstmals von Sarasvathy (2001) vorgestellt wurde, als innovativer Ansatz für unternehmerische Entscheidungsfindung an Bedeutung gewonnen. Im Gegensatz zu kausalen oder planungsorientierten Ansätzen, die auf der Vorhersage von Marktentwicklungen basieren, fokussiert die Effectuation auf die Maximierung von bestehenden Ressourcen und Möglichkeiten in unsicheren Marktbedingungen (Sarasvathy, 2001; Wiltbank et al., 2006).

Parallel dazu haben computergestützte Unternehmenssimulationen als Werkzeuge für die Modellierung und Analyse komplexer Geschäftsprozesse an Bedeutung gewonnen (Davis et al., 2007). Diese Simulationen bieten die Möglichkeit, verschiedene Geschäftsszenarien durchzuspielen und die Auswirkungen unterschiedlicher Entscheidungsparameter zu analysieren (Sterman, 2000).

Die Kombination von Effectuation und computergestützten Unternehmenssimulationen bietet ein vielversprechendes Forschungsfeld. Durch die Integration von Effectuation-Prinzipien in Simulationsmodelle könnten Unternehmen nicht nur ihre Entscheidungsprozesse in unsicheren Umgebungen verbessern, sondern auch ihre Anpassungsfähigkeit an dynamische Marktbedingungen erhöhen.

Trotz des erkennbaren Potenzials dieser interdisziplinären Herangehensweise gibt es bisher nur wenige Studien, die sich explizit mit der Verbindung von Effectuation und Unternehmenssimulationen befassen. Die meisten Simulationsmodelle in der Unternehmensforschung sind auf traditionelle, planungsorientierte Entscheidungsmodelle ausgerichtet und berücksichtigen nicht die spezifischen Herausforderungen und Möglichkeiten, die die Effectuation-Theorie bietet.

Diese Arbeit zielt darauf ab, diese Forschungslücke zu schließen. Sie soll einen Beitrag zur theoretischen und praktischen Weiterentwicklung sowohl der Effectuation-Theorie als auch der computergestützten Unternehmenssimulation leisten.

## 1.2 Problemdefinition

Während die Effectuation-Theorie ein vielversprechendes Konzept für unternehmerische Entscheidungsfindung in unsicheren Umgebungen darstellt, fehlen konkrete Werkzeuge, um diese Theorie in die Praxis umzusetzen (Chandler et al., 2011). Ebenso bieten Unternehmenssimulationen zwar die Möglichkeit, Geschäftsprozesse zu modellieren, jedoch fehlen Modelle, die die spezifischen Prinzipien der Effectuation integrieren. Die zentrale Fragestellung dieser Arbeit ist daher, wie ein computergestütztes Simulationsmodell entwickelt werden kann, das Effectuation-Prozesse im Bereich der Unternehmenssimulation abbildet.

Die moderne Unternehmenslandschaft ist von einer Vielzahl von Unsicherheitsfaktoren geprägt, die von Marktschwankungen bis hin zu technologischen Veränderungen reichen (Knight, 1921). In diesem Kontext stellt sich die Frage, wie Entscheidungsträger in Unternehmen effektive Strategien entwickeln können, um mit dieser Unsicherheit umzugehen. Traditionelle, kausale Entscheidungsmodelle, die auf der Analyse von Marktdaten und Prognosen basieren, stoßen hierbei oft an ihre Grenzen (Mintzberg, 1994).

Das Kernproblem liegt im Fehlen eines integrierten Modells, das sowohl die Prinzipien der Effectuation als auch die Komplexität realer Geschäftsprozesse berücksichtigt. Während die Effectuation-Theorie ein vielversprechendes Konzept für unternehmerische Entscheidungsfindung in unsicheren Umgebungen darstellt (Sarasvathy, 2001), fehlen konkrete Werkzeuge, um diese Theorie in die Praxis umzusetzen. Unternehmenssimulationen bieten zwar die Möglichkeit, Geschäftsprozesse zu modellieren (Davis et al., 2007), jedoch fehlen Modelle, die alle spezifischen Prinzipien der Effectuation integrieren.

Dieses Problem ist besonders relevant, da Unternehmen vor der Herausforderung stehen, in unsicheren und dynamischen Märkten zu agieren. Ein Simulationsmodell, das die Prinzipien der Effectuation integriert, könnte Unternehmen dabei unterstützen, bessere Entscheidungen zu treffen und ihre Strategien effektiver umzusetzen.

Die Herausforderungen bei der Lösung dieses Problems sind vielfältig. Erstens besteht die Herausforderung darin, die Komplexität der Effectuation-Theorie in ein quantitatives Modell zu überführen. Zweitens ist es notwendig, ein Simulationsmodell zu

entwickeln, das sowohl wissenschaftlich fundiert als auch praxisrelevant ist. Dies erfordert eine sorgfältige Auswahl und Anwendung von Methoden aus beiden Disziplinen – der Effectuation-Theorie und der Unternehmenssimulation (Read et al., 2016).

## 1.3 Abriss der zentralen Forschungsfrage und Hypothese

Die Forschungsfrage und die Hypothese dienen als Leitfaden für diese Masterarbeit und geben die Richtung der Untersuchung vor. Sie sind essenziell für die wissenschaftliche Rigorosität und die praktische Relevanz der Arbeit.

Die zentrale Forschungsfrage dieser Arbeit lautet: "Inwieweit kann ein computergestütztes Simulationsmodell entwickelt werden, um alle Effectuation-Prozesse im Bereich der Unternehmenssimulation abzubilden, und welche Möglichkeiten und Grenzen ergeben sich bei der Implementierung dieses Modells?". Diese Fragestellung ist insofern relevant, da sie die Lücke zwischen der theoretischen Fundierung der Effectuation und der praktischen Anwendbarkeit von Unternehmenssimulationen zu schließen versucht (Sarasvathy, 2001; Davis et al., 2007).

Die zugrundeliegende Hypothese der Arbeit ist: "Ein computergestütztes Simulationsmodell kann erfolgreich entwickelt werden, um Effectuation-Prozesse im Bereich der Unternehmenssimulation abzubilden, wobei die Implementierung des Modells sowohl Möglichkeiten als auch Grenzen aufweist." Diese Hypothese stellt eine prüfbare Annahme dar, die im Verlauf der Arbeit durch Modellierung und empirische Daten bestätigt oder widerlegt werden soll (Popper, 1959).

Die Forschungsfrage und die Hypothese sind nicht nur für die akademische Forschung relevant, sondern haben auch praktische Implikationen. Sie zielen darauf ab, die Lücke zwischen der Effectuation-Theorie und der Praxis der Unternehmenssimulation zu schließen, eine Lücke, die in der Literatur bisher nur wenig Beachtung gefunden hat.

Zur Beantwortung der Forschungsfrage ist es notwendig, die Schlüsselvariablen und Indikatoren zu identifizieren, die für die Überprüfung der Hypothese relevant sind. Dazu gehören unter anderem die Auswahl der Modellparameter, die Art der Datenerhebung und die Methoden der Datenanalyse (Yin, 2009). Die spezifischen Methoden, die zur Überprüfung der Hypothese angewendet werden, werden im Kapitel zur Methodik detailliert erläutert.

## 1.4 Ziele der Arbeit

Die vorliegende Masterarbeit verfolgt mehrere Ziele, die sich aus der Forschungsfrage und der Hypothese ableiten. Diese Ziele sind sowohl theoretischer als auch praktischer Natur und sollen dazu beitragen, die Lücke zwischen der Effectuation-Theorie und der Praxis der Unternehmenssimulation zu schließen (Sarasvathy, 2001; Davis et al., 2007).

Das Hauptziel dieser Arbeit ist die Entwicklung eines computergestützten Simulationsmodells, das Effectuation-Prozesse im Kontext der Unternehmenssimulation abbilden kann. Darüber hinaus sollen die Möglichkeiten und Grenzen, die bei der Implementierung dieses Simulationsmodells auftreten, untersucht werden. Weitere Ziele umfassen den Vergleich der Effectuation-Ansätze mit anderen Unternehmensstrategien innerhalb der Simulation, die Bewertung der praktischen Anwendbarkeit des entwickelten Modells in realen Unternehmensszenarien, sowie die Schaffung eines Beitrags zum aktuellen Stand der Forschung in den Bereichen Effectuation und agentenbasierte Modellierung.

Um diese Ziele zu erreichen, werden verschiedene Methoden angewendet. Dazu gehört eine umfassende Literaturrecherche, die den aktuellen Forschungsstand in den Bereichen Effectuation und agentenbasierte Modellierung ermittelt. Für die Entwicklung des Simulationsmodells werden Softwaretools und Programmiersprachen wie Python verwendet. Statistische Methoden und Datenanalysetools werden zur Auswertung der Simulationsergebnisse eingesetzt (Field, 2013). Zudem werden Interviews mit Experten aus den Bereichen Unternehmensführung und Simulation durchgeführt, um die Praktikabilität des Modells zu bewerten (Yin, 2009). Schließlich wird das entwickelte Modell auf reale Unternehmensszenarien angewendet, um seine Wirksamkeit und Anwendbarkeit zu testen (Sargent, 2013).

## 1.5 Struktur der Arbeit

Die Masterarbeit ist in sieben Hauptkapitel gegliedert, die jeweils unterschiedliche, aber miteinander verknüpfte Aspekte der Forschungsfrage und der Hypothese behandeln. Diese Struktur dient dazu, eine systematische und umfassende Untersuchung des Forschungsthemas zu ermöglichen. Im Folgenden wird ein kurzer Überblick über die Struktur und den Inhalt der einzelnen Kapitel gegeben.

**Struktur der Arbeit**

*Kapitel 2: Theoretische Grundlagen*

In diesem Kapitel wird ein fundierter Überblick über die Effectuation-Theorie und die agentenbasierte Modellierung gegeben. Es dient als theoretisches Gerüst und erläutert den aktuellen Forschungsstand in den relevanten Disziplinen.

*Kapitel 3: Methodologischer Rahmen*

Dieses Kapitel stellt die wissenschaftlichen Methoden und Analyseinstrumente vor, die für die Konzeption und Implementierung des agentenbasierten Simulationsmodells sowie für die Datenerfassung und -auswertung eingesetzt werden.

*Kapitel 4: Forschungsdesign und Konzeptionelle Überlegungen*

Dieser Abschnitt behandelt die spezifischen Fragestellungen und Herausforderungen, die als Ausgangsbasis für die Entwicklung des Simulationsmodells dienen. Er beinhaltet auch die Abgrenzung der Forschungsfragen und Hypothesen.

*Kapitel 5: Implementierung des Simulationsmodells*

In diesem Kapitel erfolgt die detaillierte Darstellung des entwickelten Simulationsmodells, einschließlich seiner theoretischen Annahmen und technischen Spezifikationen. Dies bildet den empirischen Kern der Forschungsarbeit.

*Kapitel 6: Evaluation und Ergebnisanalyse*

Die in der Simulation gewonnenen Daten werden hier präsentiert, ausgewertet und interpretiert. Das Kapitel leistet einen essentiellen Beitrag zur empirischen Forschung und adressiert die Forschungsfragen und Hypothesen.

*Kapitel 7: Schlussbetrachtungen*

Dieses abschließende Kapitel dient der kritischen Reflexion der Forschungsergebnisse. Es überprüft die formulierten Hypothesen und bietet sowohl einen Ausblick auf potenzielle zukünftige Forschungsrichtungen als auch eine abschließende Diskussion der Arbeit im Kontext des Forschungsstands.

Die Relevanz dieser Arbeit liegt in ihrer interdisziplinären Herangehensweise, die es ermöglicht, die Lücke zwischen der Effectuation-Theorie und der Praxis der Unternehmenssimulation zu schließen (Sarasvathy, 2001; Davis et al., 2007). Sie bietet nicht nur einen wissenschaftlichen Beitrag zur Weiterentwicklung der Effectuation-Theorie und der agentenbasierten Modellierung, sondern hat auch praktische Implikationen für Unternehmen, die in unsicheren und dynamischen Marktumgebungen agieren.

# 2 Theoretische Grundlagen

Dieses Kapitel dient dazu, den theoretischen Rahmen für die Forschungsarbeit zu schaffen. Es bietet eine umfassende Übersicht über die beiden Hauptthemen der Arbeit: Effectuation und agentenbasierte Modellierung. Durch das Verständnis der Grundlagen, Anwendungen und Kritikpunkte dieser Konzepte wird die Basis für die Entwicklung des Simulationsmodells gelegt, das im weiteren Verlauf der Arbeit vorgestellt wird.

## 2.1 Effectuation

Die Effectuation-Theorie ist ein bedeutender Ansatz zur Erklärung des unternehmerischen Denkens und Handelns, der von der Wissenschaftlerin Saras D. Sarasvathy entwickelt wurde (Sarasvathy, 2001). Im Gegensatz zu traditionellen, kausalen Modellen, die auf Zielsetzung, Planung und Ressourcenallokation abzielen, betont die Effectuation-Theorie den Prozess der Entscheidungsfindung und das Handeln unter Unsicherheit, wobei auf vorhandene Mittel und Ressourcen zurückgegriffen wird (Sarasvathy, 2001; Wiltbank et al., 2006).

**Grundlagen und Prinzipien**

Sarasvathy (2001) identifizierte fünf grundlegende Prinzipien der Effectuation:

### 2.1.1 „Bird in Hand“-Prinzip

Dieses Prinzip ist eines der Kernprinzipien der Effectuation-Theorie und betont die Bedeutung vorhandener Mittel und Ressourcen für unternehmerische Entscheidungen (Sarasvathy, 2001). Anstatt sich auf eine detaillierte Planung und das Erreichen bestimmter Ziele zu konzentrieren, legt das „Bird in Hand“-Prinzip nahe, dass sich Unternehmende zunächst auf das konzentrieren sollten, was sie bereits besitzen – ihre Fähigkeiten, ihr Wissen, ihre Netzwerke und andere verfügbare Ressourcen (Sarasvathy, 2001).

Indem sie sich auf diese vorhandenen Mittel fokussieren, können Unternehmende Chancen identifizieren, die zu neuen Geschäftsmöglichkeiten führen, und dabei ihre Stärken und Ressourcen effektiv nutzen (Dew et al., 2009). Dieser Ansatz ermöglicht Unternehmenden, flexibel und anpassungsfähig zu sein und auf sich ändernde Marktbedingungen oder unvorhergesehene Herausforderungen zu reagieren, ohne sich auf langfristige Pläne und Zielvorgaben zu verlassen (Sarasvathy, 2001).

Das „Bird in Hand“-Prinzip ermutigt Unternehmende, aus dem zu schöpfen, was sie bereits haben, und daraus innovative Lösungen und Geschäftsmodelle zu entwickeln, die sich an die dynamischen Gegebenheiten des Marktes anpassen können (Sarasvathy & Dew, 2005).

### 2.1.2 „Affordable Loss“-Prinzip

Dieses Prinzip der Effectuation-Theorie bezieht sich auf die Art und Weise, wie Unternehmende mit Risiken und Investitionen umgehen (Sarasvathy, 2001). Statt auf die Maximierung des erwarteten Gewinns zu fokussieren, legt das „Affordable Loss“-Prinzip nahe, dass Unternehmende den potenziellen Verlust, den sie bereit sind zu tragen, als grundlegendes Kriterium für ihre Entscheidungen betrachten (Sarasvathy, 2001).

Indem sie sich auf das „Affordable Loss“ Prinzip (in weiterer Folge auch „erschwinglicher Verlust“ bzw. „Verlusttoleranz“ genannt) konzentrieren, können Unternehmende den Umfang ihrer Investitionen und Ressourcenallokation besser einschätzen und gleichzeitig ihre Risikobereitschaft steuern (Sarasvathy, 2001). Dies ermöglicht ihnen, vorsichtig und verantwortungsbewusst mit ihren Ressourcen umzugehen, und erhöht die Wahrscheinlichkeit, ihre Geschäftsmodelle und Initiativen anpassen und verfeinern zu können, ohne dabei unverhältnismäßige Verluste zu erleiden (Sarasvathy, 2001).

Das „Affordable Loss“ Prinzip fördert einen pragmatischen Ansatz zum Risikomanagement, bei dem Unternehmende ihre Entscheidungen anhand dessen treffen, was sie bereit sind zu verlieren, anstatt sich ausschließlich auf potenzielle Gewinne und Chancen zu konzentrieren (Sarasvathy, 2001).

### 2.1.3 „Lemonade“-Prinzip

Dieses Prinzip der Effectuation-Theorie konzentriert sich darauf, wie Unternehmende mit unerwarteten Ereignissen und Unsicherheiten umgehen (Sarasvathy, 2001). Anstatt unvorhergesehene Situationen als Hindernisse oder Bedrohungen zu betrachten, ermutigt das „Lemonade“-Prinzip Unternehmende, diese als Chancen zur Anpassung und Verbesserung ihrer Geschäftsmodelle und Strategien zu sehen (Sarasvathy, 2001).

Unternehmende, die nach dem „Lemonade“-Prinzip handeln, erkennen, dass Unsicherheit und Unvorhersehbarkeit unvermeidlich sind, und suchen aktiv nach Möglich-

keiten, diese zu ihrem Vorteil zu nutzen (Read et al., 2009). Sie betrachten unerwartete Ereignisse als Möglichkeiten, neue Wege einzuschlagen, ihre Strategien zu verändern und innovative Lösungen zu finden, um ihre Geschäftsziele zu erreichen (Sarasvathy, 2001).

Das „Lemonade"-Prinzip fördert eine flexible und anpassungsfähige Herangehensweise an unternehmerische Entscheidungen und ermöglicht Unternehmenden, auf Veränderungen und Herausforderungen im Marktumfeld zu reagieren, indem sie ihre Geschäftsmodelle und Strategien kontinuierlich überprüfen und weiterentwickeln (Sarasvathy, 2001).

### 2.1.4 „Crazy Quilt"-Prinzip

Dieses Prinzip der Effectuation-Theorie betont die Bedeutung von Kooperation und Netzwerkbildung bei unternehmerischen Entscheidungen und Aktivitäten (Sarasvathy, 2001). Anstatt sich ausschließlich auf individuelle Fähigkeiten und Ressourcen zu verlassen, ermutigt das „Crazy Quilt"-Prinzip Unternehmende, Partnerschaften und Zusammenarbeit mit anderen Akteuren zu suchen, um gemeinsame Ziele und Geschäftsmöglichkeiten zu erreichen (Sarasvathy, 2001).

Durch die Bildung von Netzwerken und Kooperationen können Unternehmende ihr Wissen, ihre Fähigkeiten und Ressourcen mit denen anderer kombinieren und dadurch neue Möglichkeiten erschließen, die alle beteiligten Partner nutzen können (Read et al., 2009). Dieser Ansatz fördert eine kollektive Herangehensweise an unternehmerische Entscheidungen und gestattet Unternehmenden, gemeinsam mit anderen Akteuren auf dem Markt zu agieren und ihre Geschäftsmodelle und Strategien zu entwickeln (Sarasvathy, 2001).

Das „Crazy Quilt"-Prinzip unterstreicht die Bedeutung von Zusammenarbeit und Vernetzung im unternehmerischen Kontext und fördert eine kooperative und gemeinschaftliche Herangehensweise an die Bewältigung von Herausforderungen und die Realisierung von Geschäftsmöglichkeiten (Sarasvathy & Dew, 2005).

### 2.1.5 „Pilot in the Plane"-Prinzip

Dieses Prinzip der Effectuation-Theorie betont die Rolle der Unternehmenden als aktiver Gestalter der eigenen unternehmerischen Zukunft (Sarasvathy, 2001). Anstatt sich als Opfer äußerer Umstände und Marktbedingungen zu betrachten, ermutigt das

„Pilot in the Plane“-Prinzip Unternehmende, die Kontrolle über ihre Geschäftsentscheidungen und -aktivitäten zu übernehmen und ihre Geschäftsmodelle und Strategien proaktiv zu gestalten (Sarasvathy, 2001).

Unternehmende, die diesem Prinzip folgen, erkennen ihre Fähigkeit, Chancen zu schaffen und zu nutzen, und übernehmen die Verantwortung für ihre Entscheidungen und Handlungen (Sarasvathy, 2001). Dieser Ansatz fördert Selbstvertrauen und Eigeninitiative und ermöglicht Unternehmenden, ihre Geschäftsmodelle und Strategien entsprechend ihren individuellen Fähigkeiten, Zielen und Visionen anzupassen und weiterzuentwickeln (Sarasvathy, 2001).

Das „Pilot in the Plane“-Prinzip betont die Bedeutung von Eigenverantwortung und Selbstbestimmung im unternehmerischen Kontext und fördert einen proaktiven und zielgerichteten Ansatz zur Entwicklung und Umsetzung von Geschäftsstrategien und -modellen (Sarasvathy, 2001).

### 2.1.6 Weitere wichtige Konzepte und Parameter der Effectuation-Theorie

In der Effectuation-Theorie werden bestimmte Werte berücksichtigt, um den unternehmerischen Entscheidungsprozess und die Entwicklung von Geschäftsmodellen zu beschreiben (Sarasvathy, 2001).

*Wer bin ich?* - Die Kenntnisse, Fähigkeiten, Erfahrungen und Persönlichkeitsmerkmale der Unternehmenden beeinflussen, wie sie unternehmerische Möglichkeiten erkennen und nutzen (Sarasvathy & Dew, 2005).

*Mit wem bin ich vernetzt?* - Die Beziehungen und Netzwerke, die die Unternehmenden aufgebaut haben, beeinflussen die Ressourcen, die sie nutzen können, sowie die Geschäftsmöglichkeiten, die sich ergeben (Read et al., 2009).

*Was weiß ich?* - Das Wissen der Unternehmenden über Märkte, Technologien, Branchen und Kunden beeinflusst ihre Fähigkeit, Chancen zu identifizieren und innovative Lösungen zu entwickeln (Sarasvathy & Dew, 2005).

*Neue Mittel* - Durch den Prozess der Effectuation können Unternehmende neue Ressourcen und Fähigkeiten erschließen, die sie nutzen können, um ihre Geschäftsmodelle und Strategien zu entwickeln und umzusetzen (Sarasvathy & Dew, 2005).

*Neue Ziele* - Unternehmende können im Verlauf des Effectuation-Prozesses ihre Ziele und Visionen anpassen und verfeinern, um auf Veränderungen im Marktumfeld

zu reagieren und neue Geschäftsmöglichkeiten zu erschließen (Sarasvathy & Dew, 2005).

*Netzwerk* - Durch die Zusammenarbeit und Partnerschaften, die im Rahmen des „Crazy Quilt"-Prinzips gebildet werden, können Unternehmende ihr Netzwerk erweitern und von den Ressourcen, Kenntnissen und Fähigkeiten ihrer Partner profitieren (Sarasvathy, 2001).

*Änderung der Wissensbasis* - Im Laufe des Effectuation-Prozesses erweitern und vertiefen Unternehmende ihr Wissen über Märkte, Technologien, Branchen und Kunden, was wiederum ihre Fähigkeit zur Identifizierung und Nutzung von Geschäftsmöglichkeiten beeinflusst (Sarasvathy, 2008).

Die Konzepte der Effectuation-Theorie dienen als Grundlage für die unternehmerische Entscheidungsfindung und ermöglichen den Unternehmenden, ihre Geschäftsmodelle und Strategien an veränderte Marktbedingungen und neue Geschäftsmöglichkeiten anzupassen. Durch das Verständnis dieser Konzepte können Unternehmende ihre Ressourcen und Fähigkeiten besser nutzen und effektivere Entscheidungen treffen, um den Erfolg ihres Unternehmens zu maximieren.

### 2.1.7 Anwendungsbereiche

Effectuation hat sich als besonders nützlich in einer Reihe von Anwendungsbereichen erwiesen. In Start-up-Unternehmen beispielsweise, in denen die Unsicherheit oft hoch ist, bietet der Ansatz wertvolle Orientierung (Read et al., 2016). Auch in der Produktentwicklung und in der strategischen Entscheidungsfindung in dynamischen Branchen findet die Theorie Anwendung (Chandler et al., 2011; Sarasvathy, 2001).

### 2.1.8 Kritik und Limitationen

Trotz der breiten Anwendbarkeit und der vielen Vorteile, die Effectuation bietet, ist der Ansatz nicht frei von Kritik. Einige Forscher bemängeln beispielsweise die fehlende Struktur und die Flexibilität des Ansatzes als potenzielle Schwächen (Arend et al., 2015). Zudem ist der Ansatz in stabilen, gut verstandenen Märkten möglicherweise nicht so effektiv wie planungsorientierte Ansätze (Arend et al., 2015). Darüber hinaus kann die Anwendung der Effectuation-Theorie ressourcenintensiv sein, insbesondere in Bezug auf Zeit und soziales Kapital (Read et al., 2016).

## 2.2 Agentenbasierte Modellierung

Die agentenbasierte Modellierung (ABM) ist ein vielseitiger und flexibler Ansatz in der computergestützten Simulation, der es ermöglicht, komplexe Systeme durch die Interaktion von Agenten zu modellieren. Ein Agent ist in diesem Kontext eine autonome Einheit, die Entscheidungen trifft und mit anderen Agenten interagiert. Dieser Ansatz hat seine Wurzeln in den Sozialwissenschaften und der Informatik und wurde in den letzten Jahrzehnten in einer Vielzahl von Disziplinen angewendet.

### 2.2.1 Grundlagen und Prinzipien

Die Grundprinzipien der agentenbasierten Modellierung sind Autonomie, Interaktion, Komplexität und Adaptivität. Agenten handeln unabhängig und treffen ihre eigenen Entscheidungen, basierend auf einer Reihe von Regeln oder Heuristiken. Die agentenbasierte Modellierung ist somit besonders nützlich, um emergente Phänomene zu untersuchen, die sich aus der Dynamik von Interaktionen zwischen diesen Agenten ergeben, und gestattet Forschern, makroskopische Muster und Prozesse zu erklären, die aus mikroskopischen Wechselwirkungen entstehen (Epstein & Axtell, 1996). Sie interagieren aber nicht nur miteinander, sondern auch mit ihrer Umgebung, was zu komplexen Mustern und Phänomenen führt (Epstein & Axtell, 1996). Darüber hinaus sind Agenten in der Lage, ihr Verhalten im Laufe der Zeit anzupassen, um auf veränderte Bedingungen oder Informationen zu reagieren (Gilbert, 2008). Diese Modellierung von individuellen Unterschieden ermöglicht, die Auswirkungen von Variabilität auf das Gesamtsystem zu untersuchen und die Rolle von Agenten in der Entstehung von Mustern und Strukturen zu verstehen. Ein weiterer Vorteil von agentenbasierter Modellierung ist die Möglichkeit, räumliche und zeitliche Aspekte explizit zu berücksichtigen (Macal & North, 2010). Agenten können sich in einem Raum bewegen und ihre Interaktionen können von der räumlichen Distanz und der Zeit abhängen. Dies erlaubt die Untersuchung von räumlichen Mustern und zeitlichen Dynamiken, die in traditionellen Modellierungsansätzen möglicherweise vernachlässigt werden.

#### Parameter in agentenbasierten Simulationen

In agentenbasierten Simulationen sind die Modellparameter entscheidend, um das Verhalten und die Interaktionen von Agenten innerhalb eines Systems zu steuern und zu analysieren. Parameter können in drei Hauptkategorien eingeteilt werden: Initialisierungsparameter, Ergebnisvariablen und Simulationsparameter.

Initialisierungsparameter sind die grundlegenden Variablen, die das Modell und die Agenten definieren. Sie können einen entscheidenden Einfluss auf die Ergebnisse der Simulation haben.

Ergebnisvariablen sind die Ergebnisse und Kennzahlen, die aus der Simulation gewonnen werden. Die Analyse der Ergebnisvariablen lässt es zu, Rückschlüsse auf die Mechanismen und Prozesse innerhalb des Systems zu ziehen.

Simulationsparameter sind die technischen Parameter, die die Durchführung der Simulation beeinflussen. Diese Parameter müssen sorgfältig gewählt werden, um ein Gleichgewicht zwischen Modellkomplexität und Rechenleistung zu erreichen.

Die Kalibrierung und Validierung von Parametern in agentenbasierten Simulationen ist ein weiterer wichtiger Aspekt, der besondere Aufmerksamkeit erfordert (Thiele et al., 2014). Sensitivitätsanalysen können durchgeführt werden, um die Robustheit der Simulation gegenüber Änderungen der Parameterwerte zu überprüfen (Saltelli et al., 2008).

Eine gute Kommunikation und Dokumentation der Parameter und ihrer Annahmen ist entscheidend, um die Transparenz und Nachvollziehbarkeit der Simulation zu gewährleisten (Grimm et al., 2006). Die Verwendung von standardisierten Protokollen wie dem Overview, Design Concepts, and Details (ODD) Protokoll, kann dazu beitragen, die Beschreibung und Kommunikation von agentenbasierten Modellen zu verbessern (Grimm et al., 2010).

### 2.2.2 Anwendungsbereiche

Die agentenbasierte Modellierung wird in einer breiten Palette von Disziplinen und Anwendungsgebieten eingesetzt. In den Wirtschaftswissenschaften wird sie beispielsweise zur Modellierung von Märkten und Handelsmechanismen verwendet (Tesfatsion, 2006). In den Sozialwissenschaften hilft sie bei der Untersuchung von sozialen Netzwerken und Meinungsdynamiken (Macy & Willer, 2002). Auch in der Ökologie findet sie in der Simulation von Ökosystemen und der Interaktion zwischen verschiedenen Arten Anwendung (Grimm & Railsback, 2005). Im Gesundheitswesen wird sie zur Modellierung der Ausbreitung von Krankheiten und der Wirksamkeit von Interventionsstrategien eingesetzt (Eubank et al., 2004).

### 2.2.3 Kritik und Limitationen

Trotz ihrer Vielseitigkeit und Anwendbarkeit hat die agentenbasierte Modellierung auch ihre Grenzen. Einer der Hauptkritikpunkte ist die Rechenintensität, insbesondere wenn es um die Simulation von großen Agentenpopulationen geht (Axtell, 2000). Ein weiteres Problem ist die Kalibrierung und Validierung der Modelle (Miller & Page, 2007). Aufgrund der hohen Komplexität der Modelle ist es oft schwierig, sie anhand realer Daten zu kalibrieren oder zu validieren (Windrum et al., 2007). Zudem besteht die Gefahr der Überkomplexität, bei der Modelle so komplex werden, dass sie an Verständlichkeit und praktischer Anwendbarkeit verlieren (Edmonds & Moss, 2005).

## 2.3 Schnittstellen zwischen Effectuation und agentenbasierter Modellierung

**Bisherige Forschung und Anwendungen**

Die Verbindung von Effectuation und agentenbasierter Modellierung stellt ein aufstrebendes Forschungsfeld dar, das das Potenzial hat, sowohl die Theorie als auch die Praxis der Unternehmensführung und Entscheidungsfindung grundlegend zu beeinflussen. Obwohl beide Ansätze unabhängig voneinander entwickelt wurden, gibt es mehrere Schnittstellen, die eine fruchtbare Grundlage für interdisziplinäre Forschung bieten.

**Entscheidungsfindung unter Unsicherheit**

Sowohl Effectuation als auch agentenbasierte Modelle beschäftigen sich intensiv mit der Frage, wie Entscheidungen in unsicheren und dynamischen Umgebungen getroffen werden können. Während die Effectuation-Theorie spezifische Prinzipien und Heuristiken für die Entscheidungsfindung unter Unsicherheit bietet (Sarasvathy, 2001), ermöglichen agentenbasierte Modelle die Simulation dieser Entscheidungsprozesse in einem computergestützten Umfeld (Bonabeau, 2002).

**Komplexität und Emergenz**

Beide Ansätze erkennen die emergenten Eigenschaften komplexer Systeme an. In der Effectuation-Theorie wird die Fähigkeit der Unternehmenden hervorgehoben, komplexe und dynamische Marktbedingungen zu navigieren (Sarasvathy, 2001). Agentenbasierte Modelle bieten die Möglichkeit, diese Komplexität durch die Interaktion von Agenten zu simulieren und so ein besseres Verständnis für die Entstehung komplexer Phänomene in wirtschaftlichen Kontexten zu erzielen (Gilbert, 2008).

**Adaptive Strategien**

Die Anpassungsfähigkeit ist ein weiteres gemeinsames Merkmal beider Ansätze. In der Effectuation wird die Fähigkeit der Unternehmenden betont, ihre Strategien flexibel an veränderte Marktbedingungen anzupassen (Wiltbank et al., 2006). Agentenbasierte Modelle können diese adaptiven Strategien simulieren und bieten somit einen fruchtbaren Boden für die Erforschung der Dynamiken in unsicheren Märkten (Railsback & Grimm, 2019).

**Praktische Anwendungen**

Obwohl einige Studien bereits die Anwendung von Effectuation-Prinzipien in agentenbasierten Unternehmenssimulationen untersucht haben, bleibt dieses Feld weitgehend unerforscht und bietet zahlreiche Möglichkeiten für zukünftige Forschungen.

## 2.4 Bestehende Simulationsansätze im Kontext der Effectuation-Theorie

Die computergestützte Simulation von unternehmerischen Entscheidungsprozessen im Kontext der Effectuation-Theorie stellt ein aufstrebendes Forschungsfeld dar. In diesem Abschnitt werden zwei prominente Simulationsansätze vorgestellt und hinsichtlich ihrer Vor- und Nachteile analysiert, während ein dritter Ansatz aufgrund seiner innovativen Erweiterungen kurz erläutert wird.

### 2.4.1 Welter & Kim (2017)

Das Modell von Welter & Kim (2017) stellt einen innovativen Ansatz dar, um die Auswirkungen von Effectuation und Causation in unternehmerischen Entscheidungsprozessen unter Bedingungen von Risiko und Unsicherheit zu untersuchen. Das Modell verwendet eine agentenbasierte Simulation, die auf der NK-Fitness-Landschaft basiert, einem Konzept, das die Komplexität von Entscheidungsprozessen in Unternehmen abbildet.

In dieser simulierten Landschaft werden Unternehmen durch binäre Entscheidungsfolgen repräsentiert, die ihre "Fitness" oder Leistung bestimmen. Die Landschaft selbst kann stabil (risikobasiert) oder dynamisch (unsicher) sein, und die Unternehmen können entweder kausale oder effektuale Entscheidungsregeln verfolgen. Die Causation wird durch einen hohen Grad an Planung (hoher ρ-Wert) charakterisiert, während die Effectuation durch Flexibilität und Anpassungsfähigkeit (niedriger ρ-Wert) gekennzeichnet ist.

Ein Schlüsselparameter des Modells ist τ, der den Grad der Unsicherheit vertritt. Ein τ von 0 impliziert eine vollständig vorhersehbare Umgebung, während ein τ von 1 eine vollständig unsichere Umgebung darstellt. Ein weiterer wichtiger Parameter ist λ, der die Fähigkeit eines Unternehmens zur genauen Vorhersage des globalen Optimums repräsentiert.

Das Modell berücksichtigt auch die Interdependenz von Entscheidungen durch den Parameter K, der die Komplexität der Entscheidungslandschaft steuert. Ein höheres K führt zu einer komplexeren, "rugged" Landschaft mit mehreren lokalen Optima, während ein niedrigeres K eine glattere Landschaft mit einem einzigen globalen Optimum erzeugt.

Die Simulation folgt einem iterativen Prozess, in dem Unternehmen ihre Entscheidungen anpassen, um ihre Leistung zu verbessern. Dies geschieht durch einen "Hill-Climbing"-Mechanismus, bei dem Unternehmen versuchen, ihre aktuelle Position in der Fitness-Landschaft zu verbessern, indem sie eine Entscheidung nach der anderen ändern.

Das Modell von Welter und Kim trägt zur Literatur bei, indem es einen quantitativen Rahmen für die Untersuchung der Wirksamkeit von Effectuation und Causation unter verschiedenen Bedingungen bietet. Es ermöglicht auch eine differenzierte Betrachtung der Auswirkungen von Unsicherheit und Risiko auf unternehmerische Entscheidungsprozesse. Allerdings ist zu beachten, dass das Modell bestimmte Aspekte der Effectuation-Theorie wie den erschwinglichen Verlust und zufällige Partnerschaften nicht direkt abbildet.

### 2.4.2 Mauer et al. (2017)

Mauer et al. (2017) präsentierten ein simulationsbasiertes Modell zur Untersuchung unternehmerischer Suchprozesse, das auf der Idee basiert, dass unternehmerisches Handeln in der Schaffung neuer wirtschaftlicher Artefakte besteht, die in einem Produktmarkt finalisiert werden. Diese Produkte werden in ihrem Modell als wirtschaftliche Artefakte konzipiert, die durch Vektoren von n Komponenten repräsentiert werden, wobei jeder Vektor Produkteigenschaften darstellt.

Das Herzstück des Modells sind zwei Haupttypen von Agenten: Verbraucheragenten und Lieferantenagenten. Während Verbraucheragenten passive Rollen einnehmen, verfolgen Lieferantenagenten entweder eine vorhersage- oder kontrollbasierte Strategie zur Produktentwicklung. Vorhersageagenten sammeln und nutzen Marktinformationen, um ihre Produkte zu optimieren, während Kontrollagenten ihre Produkte

entwickeln, indem sie Stakeholder ohne Rücksicht auf vorhergesagte Marktnachfragen einbinden.

Ein weiteres zentrales Element des Modells ist die Leistungsmessung, die als Übereinstimmung eines Produkts mit der bestehenden Marktnachfrage definiert ist. Hierbei wird die Marktpassung als Prozentsatz der Verbraucheragenten berechnet, deren Nachfragevektorkomponenten mit den Produktvektorkomponenten übereinstimmen.

Darüber hinaus berücksichtigt das Modell verschiedene Aspekte der unternehmerischen Problemumgebung. Dazu gehören Faktoren wie Umweltisotropie, Umweltunvorhersehbarkeit und Zielambiguität, die alle eine Rolle bei der Gestaltung der unternehmerischen Suche spielen.

In der praktischen Anwendung wurde das Modell in drei Simulationsexperimenten getestet. Jedes dieser Experimente konzentrierte sich auf die Auswirkungen einer bestimmten Variablen auf die durchschnittliche Marktpassung der beiden verglichenen Strategien. Durch den Einsatz von Monte-Carlo-Simulationstechniken wurde sichergestellt, dass die Ergebnisse nicht durch statistische Artefakte verzerrt wurden.

Abschließend bietet das Modell von Mauer et al. (2017) einen tiefgreifenden Einblick in die Dynamik und Mechanismen unternehmerischer Suchprozesse. Es beleuchtet, wie verschiedene Strategien in unterschiedlichen Marktumgebungen agieren und wie verschiedene Faktoren die Marktpassung beeinflussen können. Dieses Modell stellt somit eine wertvolle Ressource für zukünftige Forschungen im Bereich der Unternehmensstrategie dar. In der vorliegenden Arbeit wird das agentenbasierte Simulationsmodell von Mauer et al. (2017) analysiert, das sich auf die Modellierung unternehmerischer Entscheidungsprozesse im Kontext der Effectuation-Theorie konzentriert. Während das Modell eine Reihe von wichtigen Variablen und Mechanismen integriert, darunter Umweltunsicherheit, Zielambiguität und verschiedene unternehmerische Suchstrategien, gibt es bestimmte Aspekte der Effectuation-Theorie, die nicht vollständig abgebildet werden.

Erstens scheint das Modell die Komplexität und Dynamik von Partnerschaften und Netzwerken nicht in vollem Umfang zu erfassen. Die Effectuation-Theorie legt großen Wert auf die Ko-Kreation von Möglichkeiten und die Entwicklung von Partnerschaften, die in der Simulation nur begrenzt behandelt werden.

Zweitens fehlt im Modell die explizite Berücksichtigung des „Affordable Loss“-Prinzips. In der Effectuation-Theorie werden Entscheidungen oft auf der Grundlage dessen getroffen, was man zu verlieren bereit ist, statt auf der Grundlage erwarteter Renditen. Dieses Konzept scheint im Modell nicht explizit behandelt zu werden.

Drittens konzentriert sich das Modell eher auf die Anpassung an Marktbedingungen und scheint die Rolle vorhandener Ressourcen, ein weiteres Kernkonzept der Effectuation-Theorie, nicht ausführlich zu behandeln. Das „Bird in Hand“-Prinzip, das die Nutzung vorhandener Ressourcen betont, wird nicht explizit modelliert.

### 2.4.3 Zusammenfassung

Die Analyse der Modelle von Welter & Kim (2017) und Mauer et al. (2017) zeigt, dass beide Ansätze jeweils einzigartige Stärken und Schwächen aufweisen. Während Welter & Kim eine tiefgreifende Analyse von Entscheidungsprozessen in unsicheren Kontexten ermöglichen, bietet Mauer et al. Einblicke in die Dynamik unternehmerischer Suchprozesse. Beide Modelle reflektieren jedoch nicht alle Aspekte der Effectuation-Theorie, was ihre Anwendbarkeit in bestimmten Kontexten einschränken könnte. Sterzel (2022) versucht, diese Modelle zu kombinieren und die Lücken durch die Integration von Reinforcement-Learning-Methoden zu schließen, was einen vielversprechenden Ansatz für zukünftige Forschungen darstellt.

## 2.5 Begründung für neuen Simulationsansatz

Im vorhergehenden Unterkapitel wurde eine Übersicht über existierende Modellierungsansätze und deren Vorteile sowie Unzulänglichkeiten in Bezug auf die Effectuation-Theorie präsentiert. Im folgenden Abschnitt soll nun erläutert werden, warum die vorliegende Arbeit auf einem eigens konzipierten Simulationsmodell basiert.

### 2.5.1 Vorteile des eigenen Ansatzes

Die Entscheidung, einen eigenen Modellierungsansatz zu verfolgen, anstatt auf bestehende Modelle zurückzugreifen, bietet mehrere Vorteile, die im Kontext der Effectuation-Theorie und der spezifischen Forschungsfrage dieser Arbeit besonders relevant sind:

*Parametrisierbarkeit:* Der eigene Modellansatz ermöglicht einen hohen Grad an Parametrisierbarkeit. Dies ist insbesondere nützlich, um die verschiedenen Aspekte und Prinzipien der Effectuation-Theorie in unterschiedlichen Kontexten und unter variablen Bedingungen zu untersuchen.

*Erweiterbarkeit:* Die modulare Struktur des Modells erleichtert dessen Erweiterung. Dies ist von Vorteil, wenn zusätzliche Komponenten oder Mechanismen integriert werden sollen, etwa um neue Forschungsfragen zu adressieren.

*Verständlichkeit:* Die Klarheit und Einfachheit des Modells fördern ein intuitives Verständnis der simulierten Prozesse und der zugrunde liegenden Theorien, was die Kommunikation der Forschungsergebnisse erleichtert.

*Abdeckung aller Effectuation-Konzepte:* Im Gegensatz zu vielen bestehenden Modellen, die nur bestimmte Aspekte der Effectuation-Theorie berücksichtigen, bietet der eigene Ansatz eine umfassende Integration aller relevanten Konzepte.

*Prüfung von Spezialfällen:* Die Flexibilität des Modells erlaubt die Simulation und Analyse von Spezialfällen, die in standardisierten Modellen oft nicht berücksichtigt werden können.

*Simulation individueller Agenten:* Das Modell gestattet die Simulation von individuellen Agenten mit unterschiedlichen Starteigenschaften und Strategien, was eine differenzierte Analyse der Entscheidungsfindung im Kontext der Effectuation-Theorie ermöglicht.

*Abbildung von Marktbedingungen:* Der Markt wird im Modell durch verschiedene Parameter repräsentiert, die die Marktbedingungen und deren Einfluss auf die Agenten simulieren.

*Verzögerte Belohnungen für Investitionen:* Im Einklang mit der realen Geschäftswelt sind die Auswirkungen von Investitionen nicht sofort sichtbar, sondern treten mit einer zeitlichen Verzögerung auf. Dies fügt eine zusätzliche Ebene der Komplexität sowie Realismus hinzu.

*Anpassungsfähigkeit der Agentenstrategien:* Die Agenten im Modell sind so konzipiert, dass sie ihre Strategien in jedem Zeitschritt anpassen können, was eine dynamische Interaktion und Evolution der simulierten Systeme gestattet.

*Dokumentation des Codes:* Ein weiterer Vorteil des eigenen Modellansatzes ist die umfassende Dokumentation des Codes. Dies erleichtert nicht nur die Nachvollziehbarkeit der Modellstruktur und der implementierten Algorithmen, sondern fördert auch die wissenschaftliche Transparenz und lässt eine effiziente Weiterentwicklung oder Anpassung durch andere Forscher zu.

*Anpassungsfähige Verlusttoleranz:* Im Modell ist die Verlusttoleranz der Agenten als dynamische Variable konzipiert, die sich im Laufe der Zeit anpassen kann. Dies spiegelt realistischerweise die Anpassungsfähigkeit von Unternehmen an veränderte Marktbedingungen und Risikoprofile wider.

*Endogenität der meisten Variablen:* Die meisten Variablen im Modell sind endogen, das heißt, sie werden durch das System selbst und nicht durch externe Faktoren bestimmt. Dies erhöht die Komplexität und Realitätsnähe der Simulation, da es die Wechselwirkungen zwischen den Agenten und ihrer Umgebung besser abbildet.

Durch diese und weitere nicht erwähnte Vorteile gewährleistet der eigene Modellansatz eine hohe wissenschaftliche Rigorosität und bietet gleichzeitig die Flexibilität, um die spezifischen Anforderungen und Fragestellungen dieser Forschungsarbeit zu erfüllen.

# 3 Methodologischer Rahmen

Die methodische Vorgangsweise dieser Arbeit ist darauf ausgerichtet, die Forschungsfrage systematisch und umfassend zu beantworten. Dieses Kapitel beschreibt die verschiedenen Methoden und Ansätze, die zur Datenerhebung, Analyse und Interpretation verwendet wurden, sowie Voraussetzungen zur Modellerstellung, den Pfad der Modellerstellung selbst, und die daraus folgende Beantwortung der Forschungsfrage. Es dient als Leitfaden für die Leser, um die wissenschaftliche Rigorosität und die Gültigkeit der Ergebnisse dieser Arbeit besser zu verstehen.

## 3.1 Literaturrecherche

Die Literaturrecherche ist eine kritische Komponente dieser Forschungsarbeit und dient mehreren Zwecken: Es bietet eine theoretische Grundlage, ermöglicht ein tiefgehendes Verständnis der behandelten Themen und hilft, die Forschungslücke sowie die Relevanz der Forschungsfrage zu identifizieren. Die Auswahl der Literatur erfolgt nach strengen Kriterien, die die Qualität und Bedeutsamkeit der gesamten Arbeit beeinflussen. Hierbei wird ein besonderes Augenmerk auf die Bedeutung der Werke für die Forschungsfrage und die aufgestellten Hypothesen gelegt. Zudem wird die Aktualität der Publikationen berücksichtigt, um die neuesten Erkenntnisse und Trends in der Forschung einzubeziehen. Nur Quellen von hoher wissenschaftlicher Glaubwürdigkeit, etwa aus renommierten wissenschaftlichen Zeitschriften oder von anerkannten Experten im Feld, werden in die Analyse einbezogen.

Da der Fokus dieser Arbeit auf der Effectuation-Theorie und der agentenbasierten Modellierung liegt, werden insbesondere Quellen ausgewählt, die in diesen Bereichen einen Beitrag leisten. Hierbei wurden primäre Quellen wie Sarasvathy's Originalarbeit zur Effectuation (Sarasvathy, 2001) sowie nachfolgende Studien und Anwendungen der Theorie in verschiedenen Kontexten herangezogen. Zusätzlich wurde die Literatur zu Simulationsmodellen und deren Anwendung in den Wirtschaftswissenschaften und im Unternehmertum untersucht, um Best Practices und mögliche Fallstricke bei der Modellerstellung und -anwendung zu identifizieren (Gilbert & Troitzsch, 2005).

Nach der Auswahl der Literatur erfolgte eine systematische Auswertung. Die ausgewählten Werke wurden sorgfältig gelesen und analysiert, um die Hauptthemen und -konzepte zu identifizieren. Eine kritische Analyse der Argumentationslinien und Ergebnisse der ausgewählten Literatur wurde durchgeführt, um ihre Gültigkeit und Relevanz für die Forschungsfrage zu bewerten. Diese Erkenntnisse wurden dann mit

anderen relevanten Werken verglichen, um die Position dieser Arbeit im wissenschaftlichen Diskurs zu verstehen. Schließlich wurden die gewonnenen Erkenntnisse in den Kontext der eigenen Forschung integriert.

Es ist wichtig zu betonen, dass das Literaturstudium ein fortlaufender Prozess ist. Es findet parallel zur Modellerstellung und Datenauswertung statt und wird kontinuierlich aktualisiert, um die neuesten Forschungserkenntnisse zu integrieren. Insgesamt bietet das Literaturrecherche eine robuste theoretische Grundlage für die Forschungsarbeit und ermöglicht eine fundierte Auseinandersetzung mit der Forschungsfrage und den damit verbundenen Hypothesen.

## 3.2 Informationsbeschaffung

Die Informationsbeschaffung erfolgte sowohl aus primären als auch aus sekundären Quellen. Primäre Daten wurden durch Interviews mit anonym bleiben wollenden Experten im Bereich der Unternehmenssimulation und des Unternehmertums gesammelt. Diese Interviews halfen dabei, das Simulationsmodell zu verfeinern und sicherzustellen, dass es realitätsnah und relevant für den Kontext der Forschungsfrage dieser Arbeit ist.

Sekundäre Daten wurden aus bestehenden Studien, Datenbanken und Publikationen zum Thema Effectuation und Unternehmenssimulation gesammelt. Diese Daten wurden aufbereitet und verwendet, um das Modell zu kalibrieren und zu validieren und um sicherzustellen, dass es realitätsnahe Bedingungen und Herausforderungen der Unternehmenssimulation korrekt widerspiegelt.

## 3.3 Modellerstellung

Die Modellerstellung stellt einen integralen Bestandteil dieser Forschungsarbeit zur Klärung der Forschungsfrage dar und dient somit als Fundament für die computergestützte Simulation von unternehmerischen Entscheidungsprozessen im Rahmen der Effectuation-Theorie. Im Folgenden werden die zentralen Aspekte der Modellerstellung erörtert.

**Wahl des Modellierungswerkzeugs**

Die Auswahl des geeigneten Modellierungswerkzeugs war von entscheidender Bedeutung für die Effizienz und Genauigkeit der Simulation. In dieser Arbeit wurde die Programmiersprache Python in Kombination mit einigen fertigen Bibliotheken verwendet, da sie eine robuste Umgebung für agentenbasierte Modelle bietet und eine hohe

Flexibilität in der Parametrisierung gestattet. Die Wahl fiel zudem auf Python, da sie eine breite Akzeptanz in der wissenschaftlichen Gemeinschaft genießt und umfangreiche Bibliotheken für die Berechnung, Simulation und Visualisierung bereitstellt.

**Modellparameter**

Die Modellparameter sind essenzielle Bestandteile der Simulation und beeinflussen maßgeblich ihre Ergebnisse. In diesem Modell wurden verschiedene Arten von Agenten mit jeweils eigenen Eigenschaften und Verhaltensregeln berücksichtigt. Zudem wurden Umgebungsvariablen wie Marktbedingungen und Unsicherheitsfaktoren integriert, um ein realistisches Szenario zu schaffen.

**Definition der Agenten und ihrer Eigenschaften**

Die Agenten im Modell sind als autonome Einheiten konzipiert, die jeweils eine Reihe von Eigenschaften und Verhaltensweisen aufweisen. Sie folgen unterschiedlichen Strategien zum Ressourcengewinn, wobei ihre Interaktionen durch vordefinierte Regeln gesteuert werden.

**Einbindung der Effectuation-Prinzipien**

Die Prinzipien der Effectuation-Theorie wurden in das Modell durch spezifische Entscheidungsregeln und Mechanismen integriert, die den Agenten ermöglichen, in einer unsicheren und dynamischen Umgebung effektiv zu agieren. Diese Einbindung erlaubte eine detaillierte Untersuchung der Wirkungsweise und der Effizienz der Effectuation-Prinzipien in simulierten unternehmerischen Entscheidungsprozessen.

**Kalibrierung und Validierung**

Nach der Implementierung wurde das Modell zur Sicherstellung realistischer Ergebnisse kalibriert. Dies wurde durch den Vergleich der Simulationsergebnisse mit bekannten Daten und Szenarien aus der Literatur erreicht. Die Validierung des Modells war ein fortlaufender Prozess, bei dem die Ergebnisse der Simulation mit realen und Felddaten verglichen wurden. Dieser Schritt war entscheidend, um die Genauigkeit und Relevanz des Modells für die Unternehmenssimulation des X-Pad-Projekts zu gewährleisten.

**Iterative Verfeinerung**

Basierend auf den Ergebnissen der Kalibrierung und Validierung sowie dem Feedback von Experten wurden mehrere Iterationen des Modells durchgeführt. Dieser iterative Ansatz ermöglichte, das Modell kontinuierlich zu verfeinern und sicherzustellen,

dass es die Komplexität und Dynamik der Effectuation-Theorie und der Produktrealisation korrekt abbildet.

## 3.4 Operationalisierung des Modells zur Beantwortung der Forschungsfrage

Die methodische Vorgangsweise dieser Forschungsarbeit sieht vor, das erstellte Modell als zentrales Instrument zur Beantwortung der Forschungsfrage und zur Überprüfung der Hypothese zu nutzen. Das Modell, das auf einer sorgfältigen Auswahl von Parametern, Agenteneigenschaften und Effectuation-Prinzipien basiert, wird durch computergestützte Simulationen operationalisiert. Diese Simulationen gestatten, verschiedene Szenarien und Bedingungen zu testen, um ein tiefgehendes Verständnis der untersuchten Phänomene zu erlangen. Die Ergebnisse der Modellsimulationen schaffen somit eine robuste methodische Grundlage, die gestattet, die Forschungsfrage fundiert und umfassend zu beantworten.

# 4 Forschungsdesign und konzeptionelle Überlegungen

Das Kapitel dient als integratives Fundament für die vorliegende Arbeit und verknüpft theoretische Überlegungen mit praktischen Anwendungen. Von der Klärung zentraler Arbeitsdefinitionen über die Auswahl geeigneter Arbeitsmethoden bis hin zur Darstellung der spezifischen Modellanforderungen und der Datenbasis wird in diesem Kapitel ein umfassender Rahmen für die empirische Untersuchung geschaffen. Es ist das Ziel, sowohl die wissenschaftliche Strenge als auch die praktische Relevanz der Forschung sicherzustellen. Damit legt dieses Kapitel den Grundstein für die nachfolgende Analyse und Diskussion der Forschungsergebnisse.

## 4.1 Arbeitsdefinitionen

Die Arbeitsdefinitionen dienen in dieser Arbeit als Grundlage für ein einheitliches Verständnis der Schlüsselbegriffe und Konzepte. Sie tragen dazu bei, die Forschungsfrage präzise zu formulieren und die Ergebnisse klar zu kommunizieren.

Zu den wichtigsten Begriffen, die in diesem Kontext geklärt werden müssen, gehört zunächst "Effectuation". In der Unternehmensführung und Entscheidungsfindung bezeichnet Effectuation einen Ansatz, bei dem unternehmerische Entscheidungen nicht primär auf der Basis von Vorhersagen, sondern durch den Einsatz verfügbarer Mittel und die Schaffung neuer Möglichkeiten getroffen werden. Dieser Begriff wird von verwandten Konzepten wie Kausalität und Planung abgegrenzt, die eher auf vordefinierte Ziele und systematische Analysen setzen.

Ein weiterer zentraler Begriff ist die "agentenbasierte Modellierung". Dieses Konzept bezieht sich auf die Simulation komplexer Systeme durch die Interaktion autonomer Einheiten, sogenannter Agenten. In diesem Modell repräsentieren Agenten individuelle Akteure oder Gruppen, die nach bestimmten Regeln und Strategien handeln.

Der Begriff "Unternehmenssimulation" wird ebenfalls definiert und von anderen Formen der Simulation abgegrenzt. Unternehmenssimulationen dienen speziell der Modellierung und Analyse von Geschäftsprozessen und Entscheidungsfindungen in Unternehmen. Sie sind ein wichtiges Werkzeug für Manager, um verschiedene Szenarien durchzuspielen und bessere Entscheidungen treffen zu können.

"Marktbedingungen" ist ein weiterer Schlüsselbegriff dieser Arbeit. Im Kontext der Effectuation-Theorie bezieht sich dieser Begriff auf die verschiedenen Dimensionen

des Marktes, die Unternehmende nicht kontrollieren können, aber die ihre Entscheidungen beeinflussen. Dazu gehören Faktoren wie Nachfrage, Konkurrenz und regulatorische Bedingungen.

Schließlich wird der Begriff "Entscheidungsprozesse" im Kontext von Unternehmensführung und Effectuation erläutert. Entscheidungsprozesse sind die Mechanismen und Schritte, die Unternehmende oder eine Organisation durchlaufen, um eine Wahl zwischen verschiedenen Handlungsoptionen zu treffen.

Das "X-Pad-Projekt" fungiert in dieser Forschungsarbeit als hypothetisches Unternehmensszenario, das speziell zur Demonstration der Integration der Effectuation-Theorie in den Prozess der Unternehmenssimulation konzipiert ist. In der agentenbasierten Modellierung stellt das X-Pad-Projekt einen individuellen Agenten dar, dessen Verhalten und Entscheidungsfindung im Kontext der Effectuation-Theorie analysiert werden. Dieser spezifische Agent wird im weiteren Verlauf der Studie mit dem Durchschnittsverhalten der übrigen Agenten im Simulationsmodell verglichen. Die detaillierten Charakteristika und Anforderungen des X-Pad-Projekts werden in den nachfolgenden Abschnitten der Arbeit umfassend erläutert.

Abschließend ist es von Bedeutung, den von Sarasvathy & Dew (2005) eingeführten Terminus der "Opportunity" (Geschäftsmöglichkeit) zu erläutern, da dieser direkte Implikationen für das konzipierte Simulationsmodell besitzt. Im Rahmen der Effectuation-Theorie versteht Sarasvathy Geschäftsmöglichkeiten nicht als feststehende oder objektivierbare Marktgelegenheiten, sondern als emergente Resultate eines iterativen und adaptiven Prozesses, der durch effektuale Prinzipien geleitet wird. Statt Gelegenheiten im Markt zu "entdecken", generieren Entrepreneure diese durch die Mobilisierung vorhandener Ressourcen, einschließlich Zeit und sozialer Netzwerke sowie durch Interaktionen mit diversen Stakeholdern oder Partnern. In dieser Perspektive sind Geschäftsmöglichkeiten dynamische Konstrukte, die durch die Entscheidungen und Handlungen der Entrepreneure selbst kreiert werden. Im Kontext des vorliegenden Simulationsmodells werden Geschäftsmöglichkeiten daher dualistisch konzeptualisiert: einerseits als Optionen für Investitionen oder Kapitalakkumulation und andererseits als Gelegenheiten für den Ressourcenaustausch mit potenziellen Partnern.

Durch die Klärung dieser Schlüsselbegriffe wird die terminologische Grundlage für die weitere Forschung in dieser Arbeit geschaffen.

## 4.2 Forschungsansatz und Arbeitsmethoden

Die Auswahl adäquater Arbeitsmethoden ist ein kritischer Faktor, der die wissenschaftliche Qualität und die inhaltliche Relevanz der Forschungsergebnisse maßgeblich beeinflusst. In diesem Unterkapitel werden sowohl der Forschungsansatz als auch die Arbeitsmethoden zur Datenerhebung und -analyse erläutert, um zu verdeutlichen, warum sie für die Beantwortung der Forschungsfrage und die Überprüfung der Hypothese besonders geeignet sind.

### 4.2.1 Qualitativer Ansatz

Ein qualitativer Forschungsansatz wurde gewählt, um ein tiefgehendes Verständnis der Effectuation-Theorie zu gewinnen. Zusätzlich zu einigen geführten Experteninterviews konzentrierte sich dieser Ansatz auf die Analyse von Texten, insbesondere der Primär- und Sekundärliteratur, um die sozialen Konzepte, Prinzipien und Anwendungen der Effectuation-Theorie explorativ und induktiv zu verstehen. Die qualitativen Daten boten einen reichen Kontext und ermöglichten es, Nuancen und Feinheiten in der Theorie zu erkennen, die in rein quantitativen Daten möglicherweise übersehen werden.

### 4.2.2 Quantitativer Ansatz

Parallel dazu wurde ein quantitativer Ansatz verfolgt, insbesondere bei der computergestützten Modellierung und Simulation. Dieser Ansatz gestattete, spezifische Szenarien zu simulieren und numerische Daten zu generieren, die dann statistisch analysiert werden konnten. Die quantitativen Daten boten die Möglichkeit, Muster zu erkennen, Hypothesen zu testen und spezifische Fragen zu beantworten, die in der qualitativen Analyse aufgeworfen wurden.

### 4.2.3 Begründung für den gemischten Ansatz

Die Kombination von qualitativen und quantitativen Ansätzen wurde gewählt, um die Stärken beider Ansätze zu nutzen und eine umfassende Perspektive auf die Forschungsfrage zu gewinnen. Während die qualitativen Daten ein tiefes Verständnis der Theorie und ihrer Anwendungen boten, ermöglichten die quantitativen Daten eine systematische Analyse, Bewertung der Simulationsergebnisse und Anpassung des Modells. Dieser gemischte Ansatz war besonders wichtig, um die Komplexität und Vielschichtigkeit der Effectuation-Theorie und ihrer möglichen computergestützten Simulation zu erfassen und auf das Modell abzubilden.

## 4.3 Datenerhebung und -verarbeitung

In der Datenerhebungsphase dieser Forschungsarbeit kamen primär zwei Methoden zum Einsatz: Experteninterviews und die Sammlung von Sekundärdaten aus existierenden wissenschaftlichen Publikationen. Es ist jedoch wichtig zu betonen, dass eine komplettumfassende Datenanalyse in diesem Kontext als nicht möglich angesehen wurde. Dies liegt insbesondere daran, dass die Effectuation-Theorie ein relativ junges Forschungsfeld ist und dementsprechend nur begrenzt empirische Daten zur Verfügung stehen (Perry et al., 2012). Oftmals sind die vorhandenen Daten aus Studien, die keinen direkten Bezug zur Effectuation-Theorie haben, extrapoliert, und stellen daher keine besonders zuverlässige empirische Grundlage dar.

Die Qualität und Relevanz der Daten, die in eine Simulation eingegeben werden sowie die Art und Weise, wie sie verarbeitet werden, sind entscheidend für die Gültigkeit und Zuverlässigkeit der Simulationsergebnisse. In diesem Abschnitt werden die Herkunft der Daten und die Methoden ihrer Verarbeitung detailliert beschrieben.

### 4.3.1 Datenquellen

Für dieses Simulationsmodell wurden Daten aus verschiedenen Quellen herangezogen.

*Experteninterviews:* Gespräche mit Experten auf dem Gebiet der Unternehmensgründung und Effectuation-Theorie wurden geführt, um qualitative Daten und Einsichten zu sammeln. Diese Interviews halfen, die in der Literatur gefundenen Daten zu validieren und zusätzliche nuancierte Informationen zu erhalten.

*Sekundärdaten:* Daten aus früheren Studien und Projekten, die sich mit ähnlichen Fragestellungen befasst haben, wurden ebenfalls berücksichtigt. Diese Daten boten einen historischen Kontext und ermöglichten Vergleiche mit aktuellen Trends und Entwicklungen.

### 4.3.2 Datenaufbereitung

Nachdem die Daten gesammelt wurden, war es notwendig, sie für die Simulation aufzubereiten:

*Datenbereinigung:* Alle Daten wurden zunächst auf Konsistenz, Vollständigkeit und Genauigkeit überprüft. Irrelevante oder redundante Informationen wurden entfernt und fehlende Daten wurden, wo möglich, durch Schätzungen oder Interpolationen ersetzt.

*Datenintegration:* Da die Daten aus verschiedenen Quellen stammten, war es notwendig, sie in ein einheitliches Format zu bringen, das für die Simulation geeignet ist. Dies beinhaltete die Umwandlung von qualitativen Daten in quantifizierbare Werte und die Zusammenführung von Daten aus verschiedenen Quellen zu einem kohärenten Datensatz.

*Datenanalyse:* Vor der eigentlichen Simulation wurde eine vorläufige Analyse der Daten durchgeführt, um Muster, Trends und Anomalien zu identifizieren. Dies half, die Einstellungen und Parameter der Simulation besser zu kalibrieren und sicherzustellen, dass sie realistische Szenarien widerspiegelt.

Durch diesen rigorosen Prozess der Datenerhebung und -verarbeitung wurde sichergestellt, dass das Simulationsmodell auf soliden und relevanten Daten basiert und die Ergebnisse der Simulation valide und zuverlässig sind.

## 4.4 Konzeption eines neuen Modells

In diesem Unterkapitel wird der Entwurf und die Struktur des eigens entwickelten Simulationsmodells von den Anforderungen, über die Hauptkomponenten bis hin zu Interaktionen und Attributen detailliert erläutert.

### 4.4.1 Allgemeine und spezifische Anforderungen an das Modell

Für die Konzeption eines Simulationsmodells, das die Effectuation-Theorie repräsentiert, sind sowohl allgemeine als auch spezifische Anforderungen zu berücksichtigen. Diese Anforderungen und Annahmen prägen die Architektur und Funktionalität des Modells und sind von zentraler Bedeutung für die Interpretation der resultierenden Simulationsergebnisse. In Tabelle 1 werden die allgemeinen Annahmen und Anforderungen, die das Modell leiten, detailliert dargestellt und erläutert.

Tabelle 1: Allgemeine Modellannahmen und Umsetzung

| Annahme | Beschreibung | Umsetzung im Modell |
|---|---|---|
| **Homogenität der Akteure zu Beginn** | Zu Beginn der Simulation wird angenommen, dass alle Akteure (Unternehmende) mit denselben Grundressourcen und Fähigkeiten starten. | Jeder Akteur startet mit denselben Anfangsparametern, z.B. einem bestimmten Kapital, einem Netzwerk von gleicher Größe und denselben Grundfähigkeiten. |

| | | |
|---|---|---|
| **Dynamische Umwelt** | Die Umwelt, in der die Akteure agieren, ist dynamisch und kann sich im Laufe der Simulation verändern. | Verschiedene externe Faktoren können im Laufe der Zeit variieren, z.B. Marktbedingungen, Konkurrenz oder technologische Entwicklungen. |
| **Lernfähigkeit der Akteure** | Es wird angenommen, dass Akteure aus ihren Erfahrungen lernen und ihre Strategien entsprechend anpassen können. | Nach jeder Simulationsrunde werden die Ergebnisse analysiert, und Akteure können ihre Strategien basierend auf diesen Ergebnissen anpassen. |
| **Begrenzte Rationalität** | Akteure treffen nicht notwendigerweise immer optimale Entscheidungen basierend auf den ihnen zur Verfügung stehenden Informationen und Ressourcen.. | Bei Entscheidungsprozessen werden Heuristiken und lokale Informationen verwendet, anstatt eine globale Optimierung durchzuführen. |
| **Interaktion und Kooperation** | Akteure können miteinander interagieren, kooperieren und Netzwerke bilden, um gemeinsame Ziele zu erreichen. | Es gibt Mechanismen, die es Akteuren ermöglichen, miteinander zu kommunizieren, Ressourcen auszutauschen und Partnerschaften zu bilden. |
| **Flexibilität** | Das Modell sollte verschiedene Szenarien und Bedingungen simulieren können. | Das Modell ist in der Lage, eine Vielzahl von Szenarien und Bedingungen zu simulieren. |
| **Skalierbarkeit** | Das Modell sollte in verschiedenen Größenordnungen funktionieren. | Das Modell funktioniert sowohl mit einer kleinen Anzahl von Akteuren und Zeiteinheiten als auch in großem Maßstab. |
| **Interaktivität** | Der Benutzer sollte in der Lage sein, Eingabeparameter des Modells zu ändern und die Auswirkungen dieser Änderungen in Echtzeit zu sehen. | Der Benutzer hat die Möglichkeit, sowohl Eingabe- als auch Simulationsparameter des Modells zu modifizieren und kann die unmittelbaren Auswirkungen dieser Änderungen in Echtzeit beobachten. |

| | | |
|---|---|---|
| **Datenvisualisierung** | Das Modell sollte Ergebnisse klar und verständlich darstellen können. | Das Modell ist in der Lage, Ergebnisse in einer leicht verständlichen Form darzustellen, z.B. durch Diagramme oder Tabellen. |

Tabelle 2 präsentiert die spezifischen Modellanforderungen, die sich aus der Integration der Effectuation-Theorie ergeben.

Tabelle 2: Spezifische Modellannahmen und Umsetzung

| Annahme | Beschreibung | Umsetzung im Modell |
|---|---|---|
| **„Bird in Hand"-Prinzip** | Das Modell sollte die vorhandenen Ressourcen und Fähigkeiten der Unternehmenden berücksichtigen. | Das Modell berücksichtigt die vorhandenen Ressourcen und Fähigkeiten der Unternehmenden. |
| **„Affordable Loss"-Prinzip** | Das Modell sollte die Risikobereitschaft der Unternehmenden abbilden können. | Das Modell bildet die Risikobereitschaft der Unternehmenden ab, basierend auf dem Betrag, den sie zu verlieren bereit sind. |
| **„Lemonade"-Prinzip** | Das Modell sollte unerwartete negative Ereignisse simulieren können. | Das Modell kann unerwartete negative Ereignisse simulieren und zeigt, wie Unternehmende diese in Vorteile umwandeln können. |
| **„Crazy Quilt"-Prinzip** | Das Modell sollte Partnerschaften und Kooperationen simulieren können. | Das Modell simuliert die Fähigkeit, Partnerschaften und Kooperationen zu bilden. |
| **„Pilot in the Plane"-Prinzip** | Das Modell sollte die Kontrolle und das Handeln der Unternehmenden in den Vordergrund stellen. | Das Modell stellt die Kontrolle und das Handeln der Unternehmenden in den Vordergrund. |
| **Feedback-Schleifen** | Das Modell sollte Feedback-Schleifen enthalten. | Das Modell enthält Feedback-Schleifen, um zu zeigen, wie Unternehmende aus Erfahrungen lernen und ihre Strategien anpassen. |

| | | |
|---|---|---|
| **Unsicherheit und Dynamik** | Das Modell sollte die unsichere und dynamische Natur des Unternehmertums abbilden können. | Das Modell kann die unsichere und dynamische Natur des Unternehmertums abdecken, z.B. durch zufällige Ereignisse oder Marktveränderungen. |
| **Ressourcenallokation** | Das Modell sollte die Allokation und Reallokation von Ressourcen simulieren können. | Das Modell simuliert die Allokation und Reallokation von Ressourcen, basierend auf den Entscheidungen und Handlungen der Unternehmenden. |
| **Netzwerkbildung** | Das Modell sollte soziale Netzwerke und Beziehungen simulieren können. | Das Modell hat die Fähigkeit, soziale Netzwerke und Beziehungen zwischen Unternehmenden und anderen Akteuren zu simulieren. |

Diese Anforderungen dienen als Leitfaden für die weiteren Modellierungsaktivitäten und stellen sicher, dass das Modell die Forschungsziele adäquat unterstützt.

### 4.4.2 Technische und konzeptuelle Anforderungen

**Technische Anforderungen**

*Softwareplattform:* Die Wahl fiel auf die Programmiersprache Python als geeignete Basis für die Implementierung des Modells. Eine ausführliche Begründung und Diskussion dieser Entscheidung wird in Kapitel 5.2.1 präsentiert.

*Hardwareanforderungen:* Für die Ausführung des Modells sind keine spezialisierten Hardwarekomponenten erforderlich. Ein X-86-basierter Computer mit einer vorinstallierten Python-Umgebung ist ausreichend, um die Simulation durchzuführen.

*Skalierbarkeit:* Das entwickelte Modell zeichnet sich durch seine nahtlose Skalierbarkeit aus. Es ist jedoch zu beachten, dass mit zunehmender Komplexität des Modells die Hardwareanforderungen und die Ausführungszeit proportional ansteigen könnten.

**Konzeptuelle Anforderungen**

Zu den konzeptuellen Anforderungen des Modells gehört die vollständige Integration aller Prinzipien der Effectuation-Theorie, diese ist in Tabelle 3 detailliert dargelegt.

Tabelle 3: Abbildung der Effectuation-Prinzipien auf das Modell

| Prinzip | Abbildung im Modell |
|---|---|
| **„Bird in Hand“-Prinzip** | Agenten verfügen über verschiedenartige Ressourcen und orientieren ihre Strategien basierend auf diesen kontinuierlich neu. Diese Optimierung der eigenen Ressourcen gestattet den Agenten auch bei unvorhergesehenen Ereignissen handlungsfähig zu bleiben. |
| **„Affordable Loss“-Prinzip** | Die Verlusttoleranz erfolgt als eine Abbildung durch einen speziellen Parameter, der für jeden Agenten individuell festgelegt wird. Diese wird durch den Agenten in jedem Zeitschritt dynamisch angepasst, je nach Marktlage, ob im vorherigen Zeitschritt Gewinn oder Verlust erwirtschaftet wurde, und ob andere Agenten eine Geschäftsmöglichkeit verfolgt haben oder nicht. |
| **„Lemonade“-Prinzip** | Repräsentation dieses Mechanismus erfolgt durch den Parameter "Kontrollfaktor". Dieser Parameter wird dynamisch angepasst und berücksichtigt den Erfolg vorangegangener Ressourcenaustauschaktionen eines Agenten. Der Kontrollfaktor dient dazu, die Auswirkungen unvorhersehbarer Ereignisse zu modulieren, indem er die Fähigkeit des Agenten erhöht, solche Ereignisse entweder abzumildern oder sogar in positive Entwicklungen umzuwandeln. |
| **„Crazy Quilt“-Prinzip** | Ein Agent hat die Möglichkeit, sein soziales Netzwerk durch die Etablierung von Partnerschaften mit anderen Agenten zu erweitern. Diese Beziehungsstrukturen dienen als Plattform für den Ressourcenaustausch, wodurch der Agent sowohl seine eigenen Bedarfe adressieren als auch die Bedarfe der Partneragenten befriedigen kann. |
| **„Pilot in the Plane“-Prinzip** | Als „Pilot“ richtet jeder Agent in jedem Zeitschritt seine Strategien neu aus, basierend auf einer Kombination aus seinen vorhandenen Ressourcen, der aktuellen Marktlage, und den aus vorangegangenen Zeitschritten akkumulierten Erfahrungen. |

### 4.4.3 Hauptkomponenten des Modells

Im folgenden Abschnitt werden die Hauptkomponenten des entwickelten Simulationsmodells grundlegend vorgestellt. Diese Komponenten bilden das strukturelle Gerüst

und sind entscheidend für die Abbildung der Dynamiken und Interaktionen, die im Rahmen der Effectuation-Theorie untersucht werden. Zu den Hauptkomponenten zählen insbesondere die Agenten, die die handelnden Einheiten im Modell repräsentieren, und die Umgebung, die den Kontext für die Entscheidungsfindung und Interaktion bietet.

**Agenten**

*Entscheidungsträger:* Jeder Akteur im Modell trifft Entscheidungen basierend auf den ihm zur Verfügung stehenden Ressourcen, seinem aktuellen Wissen, seinen Zielen und den Prinzipien der Effectuation. Diese Entscheidungen können sich auf Investitionen, Partnerschaften, Marktstrategien oder andere geschäftsrelevante Aktivitäten beziehen.

*Träger von Ressourcen:* Akteure besitzen und verwalten verschiedene Ressourcen, darunter finanzielle, humane und soziale. Diese beeinflussen ihre Handlungsfähigkeit und ihre Entscheidungsfindung im Geschäftsprozess.

*Interaktive Einheiten:* Akteure interagieren nicht nur mit ihrer Umgebung, sondern auch miteinander. Diese Interaktionen können kooperativ (z.B. Partnerschaften, Joint Ventures) oder wettbewerbsorientiert (z.B. Konkurrenz um Marktanteile) sein.

*Lernende Einheiten:* Akteure im Modell sind nicht statisch. Sie lernen aus ihren Erfahrungen, aus den Ergebnissen ihrer Entscheidungen und aus den Interaktionen mit anderen Akteuren. Dieses Lernen beeinflusst ihre zukünftigen Entscheidungen und Handlungen.

*Anpassungsfähige Einheiten:* Als Reaktion auf Veränderungen in der Umwelt oder auf dem Markt können Akteure ihre Strategien und Handlungen anpassen. Dies zeigt ihre Fähigkeit zur Flexibilität und Anpassungsfähigkeit, die im Effectuation-Prozess entscheidend ist.

*Repräsentanten von Unsicherheit:* Ein zentrales Merkmal der Effectuation-Theorie ist die Betonung von Unsicherheit. Akteure im Modell stehen ständig vor Unsicherheiten, sei es in Bezug auf Marktbedingungen, das Verhalten anderer Akteure oder die Ergebnisse ihrer eigenen Entscheidungen. Ihre Fähigkeit, mit dieser Unsicherheit umzugehen und sie sogar zu ihrem Vorteil zu nutzen, ist ein Schlüsselelement ihrer Rolle.

**Umgebung**

*Marktlage:* Dieser schwankende Wert repräsentiert die allgemeinen Marktbedingungen, die von den kollektiven Aktionen der Akteure beeinflusst werden.

*Unerwartete Ereignisse:* Diese zufälligen Ereignisse können den Markt beeinflussen und erfordern, dass Akteure reagieren und ihre Strategien anpassen.

### 4.4.4 Hauptattribute, Interaktionen und Mechanismen des Modells

In diesem Abschnitt werden die Interaktionen zwischen den Akteuren, ihre Hauptattribute und die Mechanismen, die im Simulationsmodell implementiert wurden, grundlegend beschrieben.

**Hauptattribute**

In diesem Simulationsmodell sind alle Attribute, insbesondere die Ressourcen, abstrahiert, um eine vereinfachte, aber dennoch aussagekräftige Darstellung der realen Unternehmensdynamik zu ermöglichen.

*Finanzielle Ressourcen:* Alle Akteure beginnen die Simulation mit einem festgelegten Kapitalbetrag, der für verschiedene Aktivitäten wie Investitionen, Partnerschaften oder den Erwerb von anderen Ressourcen genutzt werden kann. Dieser ist über die Dauer der Simulation volatil und kann durch verschiedene Ereignisse beeinflusst werden. Dieses Kapital kann als Sicherheit dienen, um Partnerschaften oder Kooperationen zu etablieren und beeinflusst direkt die Handlungsfähigkeit der Akteure. Ein unzureichendes Kapital kann die Möglichkeiten der Akteure einschränken, während ein Überschuss an Kapital zusätzliche Chancen eröffnen kann. Wenn Akteure ihr finanzielles Kapital aufbrauchen, scheiden sie aus der Simulation aus.

*Humane Ressourcen:* Die humanen Ressourcen im Modell umfassen die Fähigkeiten, das Wissen und die Erfahrung der Akteure. Sie repräsentieren beispielsweise ihr technisches Wissen, Marktkenntnisse und Managementfähigkeiten. Diese Fähigkeiten und das Wissen beeinflussen die Entscheidungsfindung und Handlungsfähigkeit der Akteure. Akteure mit hohem technischem Wissen könnten beispielsweise besser in der Lage sein, ihr Wissen gegen finanzielle Ressourcen zu tauschen. Zudem können sie ihr Wissen durch Lernen und Erfahrung erweitern.

*Soziale Ressourcen:* Schließlich beziehen sich die sozialen Ressourcen auf die Beziehungen, Netzwerke und Verbindungen, die Akteure in der Geschäftswelt pflegen. Das Modell berücksichtigt die Größe des Netzwerks der Akteure, das angibt, wie viele andere Akteure sie direkt kennen und mit denen sie potenziell für einen Ressourcenaustausch interagieren können. Ein umfangreiches Netzwerk kann Akteuren Vorteile

in Form von Zugang zu zusätzlichen Ressourcen, Informationen oder Geschäftsmöglichkeiten bieten, und schützen sie vor den Auswirkungen unerwarteter Ereignisse. Das Aufbauen und Pflegen eines starken Netzwerks können auch das Vertrauen und die Reputation der Akteure in der Geschäftswelt stärken.

*Verlusttoleranz:* Dieser Parameter stellt einen zentralen Wert dar, der die Toleranz der Akteure gegenüber Verlusten bei Investitionen oder Ressourcenaustausch quantifiziert. Er dient als Maßstab für die Risikobereitschaft und die strategische Ausrichtung der Agenten im Modell.

*Kontrollfaktor:* Der Kontrollfaktor ist ein quantitativer Indikator, der die Fähigkeit der Akteure repräsentiert, sich an unerwartete Ereignisse anzupassen. Er misst nicht nur die Resilienz der Agenten gegenüber unvorhersehbaren Marktveränderungen, sondern auch ihre Fähigkeit, aus solchen Ereignissen potenzielle Vorteile zu generieren.

*Marktlage:* Dieser Parameter charakterisiert den Zustand des Marktes und beeinflusst sowohl die Höhe des Ressourcenaustauschs zwischen den Agenten als auch die potenziellen Erlöse aus Geschäftsmöglichkeiten. Er dient als externer Faktor, der die Interaktionsdynamiken und Entscheidungsprozesse der Agenten im Modell moduliert.

**Interaktionen**

*Kooperation und Partnerschaft:* Akteure können miteinander kooperieren, um gemeinsame Ziele zu erreichen. Diese Kooperationen können in Form von Partnerschaften, Joint Ventures oder anderen Geschäftsbeziehungen auftreten. Akteure können versuchen, andere Akteure als Partner zu gewinnen. Partnerschaften sind vonnöten, um einen Ressourcenaustausch vollführen zu können.

*Aktionen:* Akteure können sich, sofern es ihre Verlusttoleranz es erlaubt, basierend auf ihren Ressourcen und dem Marktumfeld pro Zeitschritt für eine Aktion, auch Geschäftsmöglichkeit genannt, entscheiden. Dabei können sie zwischen „investieren“ oder „sparen“ wählen. Die Auswirkungen setzen zeitverzögert ein. Eine derartige Handlung kann als Geschäftsmöglichkeit konzeptualisiert werden, da sie das Potenzial bietet, Ressourcen zu akkumulieren. „Sparen“ akkumuliert sehr wenig Ressourcen, „investieren“ ist hingegen risikobehaftet mit potentiell höheren Erträgen, jedoch möglicherweise auch Verlusten.

*Ressourcenaustausch:* Sollten gewisse Kriterien erfüllt sein, können Akteure finanzielle oder Humanressourcen miteinander austauschen, um ihre individuellen oder ge-

meinsamen Ziele zu erreichen. Eine derartige Handlung kann ebenso als Geschäftsmöglichkeit konzeptualisiert werden, da sie je nach Parametrisierung auch das Potenzial bietet, Ressourcen zu akkumulieren.

*Feedback und Lernen:* Nach jeder Simulationsrunde erhalten die Akteure Feedback über ihre Performance. Dieses Feedback dient als Grundlage für das Lernen und die dynamische Anpassung ihrer Strategien.

**Mechanismen**

*Strategische Ausrichtung:* Dies bezieht sich auf die bevorzugte Vorgehensweise der Akteure. Ihre strategische Ausrichtung wird in jedem Zeitabschnitt dynamisch, basierend auf ihren vorhandenen und benötigten Ressourcen sowie ihren Lernfähigkeiten, angepasst.

*Marktdynamik:* Der Markt, in dem die Akteure agieren, ist dynamisch. Dieser Mechanismus stellt sicher, dass sich Marktbedingungen, wie Nachfrage und Angebot im Laufe der Zeit ändern können.

*Unerwartete Ereignisse:* Um die Realität der Geschäftswelt, in der unvorhergesehene Ereignisse auftreten können, widerzuspiegeln, sind zufällige Ereignisse in die Simulation integriert. Dies könnte beispielsweise ein plötzlicher Technologiewandel oder ein unerwarteter Markteinbruch sein.

*Lernalgorithmus:* Basierend auf dem Feedback nach jeder Runde passt dieser Mechanismus die Strategien und Aktionen der Akteure an. Dies ermöglicht den Akteuren, aus ihren Erfahrungen zu lernen und ihre Vorgehensweise entsprechend zu ändern.

*Netzwerkbildung:* Akteure können aktiv versuchen, ihr Netzwerk zu erweitern, indem sie neue Geschäftsbeziehungen und Partnerschaften eingehen. Dieser Mechanismus gestattet den Akteuren, ihre Reichweite und Ressourcenbasis zu erweitern, sowie auf unvorhergesehene Ereignisse besser vorbereitet zu sein.

*Verlusttoleranz:* In der Effectuation-Theorie wird die Verlusttoleranz als die Menge an Ressourcen definiert, die Unternehmende zu verlieren bereit sind, um unvorhersehbare Chancen zu nutzen. Dieser Ansatz betont die Bedeutung des Setzens von erschwinglichen Verlusten statt potenziellen Erträgen, um das Unternehmensrisiko zu steuern und zu minimieren.

Durch die sorgfältige Einstellung und Kombination dieser Parameter und Mechanismen kann das Simulationsmodell verschiedene Szenarien und Umgebungen abbilden, die den realen Bedingungen von Unternehmenden und ihren Entscheidungsprozessen entsprechen. Dies lässt auch zu, Hypothesen über das Verhalten von Akteuren unter verschiedenen Bedingungen zu testen und zu validieren.

## 4.5 Modellaufbau

Dieses Modellkonzept zielt darauf ab, eine Synthese der zuvor genannten Anforderungen zu schaffen und gleichzeitig ein sowohl einfaches als auch erweiterbares Grundgerüst bereitzustellen. In diesem Rahmen werden die Kernprinzipien der Effectuation-Theorie implementiert und operationalisiert. Im Weiteren wird ein Simulationsmodell präsentiert, dessen Struktur in Abbildung 1 abstrahiert dargestellt ist. Die genaue Beschreibung der Implementierung folgt in Kapitel 5.

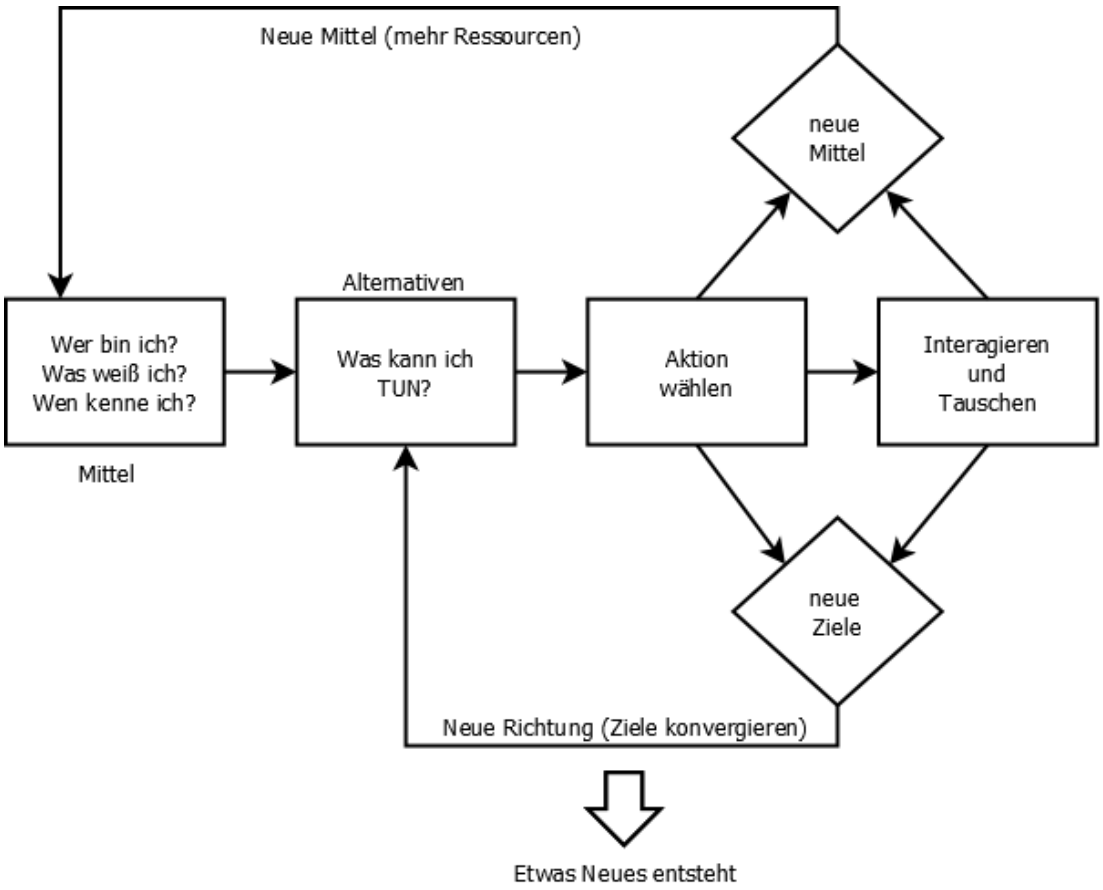


Abbildung 1: Eigene Modellstruktur angelehnt an Sarasvathy & Dew (2005)

## 4.6 Validierung des Modells

*Interne Validierung:* Im Bereich der internen Validierung wurde besonders Wert auf Sensitivitätsanalysen und die Replikation von Experimenten innerhalb des Simulationsmodells gelegt. Das Modell verfügt über eine integrierte Funktion, die es ermöglicht, Experimente mehrfach durchzuführen und statistische Kennzahlen wie Durchschnittswerte und Standardabweichungen für alle Hauptparameter zu generieren. Diese Funktion bietet zudem die Möglichkeit, spezifische Komponenten des Modells temporär zu deaktivieren. Dies dient der Überprüfung der Integrität des Modells und gestattet, die Auswirkungen einzelner Funktionen isoliert zu betrachten.

*Externe Validierung:* Im Kontext der externen Validierung war die Durchführung nur in rudimentärem Umfang möglich, was hauptsächlich auf den noch jungen Forschungsstand der Effectuation-Theorie zurückzuführen ist, wie bereits in Kapitel 4.3 erläutert. In diesem Zusammenhang wurde primär auf die Extrapolation von Daten aus anderen Forschungsbereichen zurückgegriffen, um diese den Ergebnissen des entwickelten Modells gegenüberzustellen.

Um eine umfassende Validierung dieses sowie weiterer, bestehender Simulationsmodelle zu gewährleisten, ist die zukünftige Sammlung und Auswertung zusätzlicher, empirisch robuster Daten erforderlich.

## 4.7 Zusammenfassung

In diesem Kapitel wurde das Forschungsdesign und die Umsetzungskonzeption ausführlich dargelegt und somit der Arbeitsrahmen für die Forschung definiert. Der Fokus lag auf der Modellierung eines computergestützten Simulationsmodells, das die Effectuation-Theorie abbildet, um dessen Anwendbarkeit im Kontext der Unternehmenssimulation, speziell für das X-Pad-Projekt, zu bewerten. Der gewählte Forschungsansatz wurde erläutert, und die Konzeption eines neuen Modells dargelegt.

Die Annahmen und Hauptkomponenten des Modells wurden detailliert beschrieben, einschließlich der abstrahierten Attribute wie Ressourcen und Akteure. Die Simulationsparameter wurden grundlegend festgelegt, und die Methoden für die Datenerhebung und -verarbeitung erläutert.

Dieses Kapitel dient als Grundlage für die anschließende Implementierung des Modells, die im nächsten Kapitel präsentiert wird.

# 5 Implementierung des Simulationsmodells

Dieses Kapitel konzentriert sich auf die technischen und methodischen Aspekte der Implementation des in dieser Arbeit entwickelten Modells. Es dient als Brücke zwischen der theoretischen Konzeption und der praktischen Umsetzung des Simulationsmodells, das die Effectuation-Theorie im Kontext einer Unternehmenssimulation abbildet. Zu Beginn wird das X-Pad-Projekt vorgestellt, das nun als Akteur mit voreingestellten Ressourcen fungiert, und eine Prüfmaßnahme des Simulationsmodells darstellt. Anschließend wird die technische Umsetzung des Modells erläutert, und abschließend werden die Limitationen der Implementierung diskutiert, um ein vollständiges Bild der Stärken und Schwächen des entwickelten Modells zu vermitteln.

## 5.1 Konzeptunternehmen X-Pad-Projekt

Das X-Pad-Projekt dient als Konzeptunternehmen, das eine hypothetische Produktidee, nämlich das X-Pad, entwickelt hat. Dieses innovative Eingabegerät zielt darauf ab, die Mensch-Computer-Interaktion durch eine Reihe von spezialisierten Funktionen und Gestensteuerungen zu revolutionieren. Das Konzeptunternehmen ist nicht nur ein theoretisches Konstrukt für die Produktentwicklung, sondern auch ein integraler Bestandteil eines Simulationsmodells, in dem es als Agent mit spezifischen, vordefinierten Ressourcen agiert.

Innerhalb eines agentenbasierten Simulationsmodells fungiert das X-Pad-Projekt als ein spezialisiertes Beispiel für einen Unternehmensagenten. Die speziellen Ressourcen und Fähigkeiten dieses Agenten, die aus der konkreten Produktidee des X-Pads abgeleitet sind, dienen als Startkonfiguration für die Simulation. Die zugrunde liegende Hypothese ist, dass unterschiedliche Startkonfigurationen der Agenten zu variablen Ergebnissen innerhalb der simulierten Unternehmensumgebung führen können.

Die Rolle des X-Pad-Projekts im Simulationsmodell ist mehrschichtig. Einerseits dient es als Testfall für die Wirksamkeit und Anwendbarkeit des Modells, insbesondere in Bezug auf die Fähigkeit, die Dynamik und Anpassungsfähigkeit von Start-up-Unternehmen zu erfassen. Andererseits bietet es eine Plattform für die Untersuchung, wie spezialisierte Ressourcen und Fähigkeiten die Entscheidungsfindung und Strategieentwicklung eines Unternehmensagenten beeinflussen können.

In diesem Kontext wird das X-Pad-Projekt auch als Mechanismus zur Untersuchung der Effectuation-Theorie verwendet. Durch die Simulation können verschiedene

Effectuation-Prinzipien wie das „Bird in Hand“-Prinzip, das „Affordable Loss“ Prinzip und das „Crazy Quilt“-Prinzip in einem kontrollierten Umfeld getestet werden. Die Ergebnisse könnten dazu beitragen, das Verständnis für die Anwendbarkeit dieser Prinzipien in der realen Unternehmenswelt zu vertiefen.

Im abschließenden Simulationsmodell wird das X-Pad-Projekt zur Vereinfachung der Darstellung als Agent 100 repräsentiert.

## 5.2 Technische Umsetzung

In diesem Abschnitt wird der Ablauf sowie die Implementierung des Simulationsmodells beschrieben, das auf der Effectuation-Theorie basiert. Das Modell wurde in der Programmiersprache Python entwickelt und nutzt die *Mesa*-Bibliothek für agentenbasierte Simulationen sowie *NumPy* für numerische Operationen, und erweiterte Visualisierungsbibliotheken wie *matplotlib*.

### 5.2.1 Softwarearchitektur und technologische Grundlagen

Die Architektur des Modells ist modular aufgebaut, um eine einfache Anpassung und Erweiterung zu ermöglichen. Das Modell besteht aus verschiedenen Komponenten, die jeweils für bestimmte Aspekte der Simulation verantwortlich sind, wie z.B. die Verwaltung von Ressourcen, die Interaktion zwischen Akteuren und die Implementierung der Effectuation-Prinzipien.

Python wurde als Programmiersprache für dieses Modell aus mehreren Gründen ausgewählt. Erstens ist Python eine weit verbreitete Sprache in der wissenschaftlichen Gemeinschaft, insbesondere in den Bereichen Data Science und maschinelles Lernen. Dies ist auf seine einfache Syntax, seine umfangreichen Bibliotheken und Frameworks und seine starke Unter-stützung für numerische und wissenschaftliche Berechnungen zurückzuführen.

Zweitens bietet Python eine hohe Lesbarkeit und eine intuitive Syntax, die es ideal für akademische Projekte macht, bei denen der Code oft von anderen Forschern gelesen und verstanden werden muss. Python-Code ist oft viel kürzer und einfacher zu lesen als Code in anderen Sprachen, was die Wartung und Überprüfung des Codes erleichtert.

Drittens ist Python eine interpretierte Sprache, was bedeutet, dass der Code Zeile für Zeile ausgeführt wird. Dies macht es ideal für explorative Analysen und interaktive

Modellierung, da Änderungen am Code sofort wirksam werden und nicht kompiliert werden müssen.

Viertens hat Python eine aktive und hilfsbereite Gemeinschaft von Entwicklern, wodurch es eine Fülle von Ressourcen, Tutorials und Foren gibt, um bei der Lösung von Problemen zu helfen. Darüber hinaus gibt es eine Vielzahl von Bibliotheken und Frameworks wie *NumPy* und *Pandas*, die speziell für wissenschaftliche Berechnungen, Datenanalyse und maschinelles Lernen entwickelt wurden.

Schließlich ist Python plattformübergreifend kompatibel, es kann auf jedem Betriebssystem ausgeführt werden kann. Dies ist besonders nützlich für die Zusammenarbeit zwischen Forschern, die unterschiedliche Systeme verwenden.

*Mesa* ist ein agentenbasiertes Modellierungsframework in Python. Es bietet die Werkzeuge, um agentenbasierte Simulationen zu erstellen und auszuführen. Im vorgestellten Modell dient *Mesa* zur Implementierung der Agenten und ihrer Interaktionen, gemäß den Prinzipien der Effectuation-Theorie.

Die *NumPy* Bibliothek bietet Unterstützung für große, mehrdimensionale Arrays und Matrizen sowie eine breite Palette von mathematischen Funktionen, um diese zu manipulieren. Im implementierten Modell wird *NumPy* für die effiziente Handhabung von Datenstrukturen und Berechnungen verwendet.

### 5.2.2 Allgemeine Modellarchitektur

In diesem Modell wird das unternehmerische Ökosystem durch eine Reihe von Agenten, auch Akteure genannt, dargestellt, die als Unternehmen fungieren. Das Herzstück der Simulation sind zwei Hauptklassen: EntrepreneurAgent und EntrepreneurModel.

#### Implementierungsansatz

Die Implementierung folgt einem objektorientierten Ansatz, um die Komplexität zu bewältigen und die Wiederverwendbarkeit des Codes zu fördern. Methoden innerhalb der EntrepreneurAgent-Klasse wie `choose_action` oder `interact_and_exchange`, sind dafür verantwortlich, die Logik der Entscheidungsfindung und die Dynamik der Ressourcen zu handhaben.

Die Implementierung des Modells wurde iterativ durchgeführt. Nach jeder Iteration fand eine Überprüfung der Funktionalität und der Übereinstimmung mit den theoretischen Grundlagen der Effectuation-Theorie statt. Dieser iterative Ansatz ermöglichte,

Fehler frühzeitig zu erkennen und Anpassungen vorzunehmen, um die Validität des Modells zu gewährleisten.

**EntrepreneurAgent (Agent)**

Die Klasse EntrepreneurAgent repräsentiert die Unternehmenden als Agenten im Modell. Jeder dieser Agenten ist mit einer Reihe von Ressourcen ausgestattet, die in drei Kategorien unterteilt sind: finanzielle Ressourcen, menschliche Ressourcen und soziale Ressourcen. Finanzielle Ressourcen repräsentieren das Kapital, das den Unternehmenden zur Verfügung steht, menschliche Ressourcen repräsentieren das Wissen, die Fähigkeiten und Erfahrungen der Unternehmenden und soziale Ressourcen repräsentieren das Netzwerk der Unternehmenden, das aus Beziehungen zu anderen Agenten, sprich Unternehmen, besteht. Diese Attribute sind abstrakte Repräsentationen der realen Ressourcen und Fähigkeiten, die Unternehmende in der realen Welt besitzen würden. Sie sind zentral für die Entscheidungsfindung im Rahmen der Effectuation-Theorie, und ändern sich über die Simulationszyklen hinweg.

Ein weiteres signifikantes Attribut beschreibt die Verlusttoleranz, einen variablen Parameter, der die Risikobereitschaft eines Agenten in Bezug auf Investitionen oder Ressourcenaustausch repräsentiert. Diese Verlusttoleranz, welche das „Affordable Loss“ Prinzip nach Sarasvathy (2001) abbildet, passt sich auch dynamisch im Zeitverlauf an. Die Verlusttoleranz wird durch eine Reihe von Variablen moduliert, darunter die aktuelle Marktsituation, Fluktuationen im Ressourcenbestand sowie die Anzahl der Agenten, die eine Geschäftsmöglichkeit aktiv verfolgen.

Diese zuvor genannten Attribute sind bei der Initialisierung des Modells standardmäßig fixiert. Sie können jedoch durch die Modifikation einer spezifischen Variablen dynamisch randomisiert werden, um sicherzustellen, dass jeder Agent mit einer individuellen Ausgangskonfiguration in die Simulation eintritt.

Bedeutsam ist ebenso das Attribut des Kontrollfaktors, der die Fähigkeit eines Agenten zur Anpassung an unerwartete Ereignisse quantifiziert. Ein erhöhter Wert des Kontrollfaktors deutet auf eine gesteigerte Resilienz gegenüber den negativen Konsequenzen unvorhersehbarer Ereignisse hin und besitzt das Potenzial, diese negativen Auswirkungen unter bestimmten Umständen sogar in positive Resultate zu transformieren. Der Kontrollfaktor fungiert als operationalisierte Darstellung des von Sarasvathy formulierten „Lemonade“-Prinzip, das die Kapazität zur Kontrolle und Anpassung in unsicheren Kontexten adressiert. Der Kontrollfaktor wird automatisch als das

Verhältnis der Anzahl der Agenten, mit denen primär erfolgreiche Ressourcenaustauschaktionen durchgeführt wurden, zur Gesamtzahl der interagierenden Agenten berechnet.

**EntrepreneurModel (Umgebung)**

Die Klasse EntrepreneurModel dient als das übergeordnete Modell, das die Interaktionen zwischen den Agenten steuert und die Simulation als Ganzes verwaltet. Es initialisiert die Agenten, setzt Simulationsparameter wie Marktbedingungen und führt die Simulationsschritte aus, in denen die Agenten ihre Entscheidungen treffen, ihre Ressourcen anpassen und ihre Netzwerke verwalten. In jedem Simulationszyklus bietet das Modell die Möglichkeit, Agenten mit unzureichenden Ressourcen aus dem System zu eliminieren. Dies dient der Abbildung der inhärenten Dynamik im unternehmerischen Ökosystem. In diesem Modell wurde bewusst darauf verzichtet, neue Agenten in den Markt eintreten zu lassen. Dieser Verzicht dient der Stabilität der Ausgabeparameter, um deren Aussagekraft nicht zu beeinträchtigen.

Darüber hinaus ist die Umgebung für die Generierung unerwarteter Ereignisse verantwortlich. Aufgrund der inhärenten Komplexität realer Bedingungen werden diese Ereignisse im Modell in einer abstrahierten und exemplarischen Weise dargestellt. Zusätzlich eliminiert die Umgebung Agenten, die am Ende eines Zeitschritts über unzureichende Ressourcen verfügen.

Die Marktbedingungen, welche ein Attribut der Umgebung darstellen, werden in diesem Modell durch die kollektiven Handlungen der Agenten bestimmt, genauer gesagt durch die Summe der Verhältnisse investierender Agenten zu allen Agenten und erfolgreicher Ressourcenaustauschs zu allen Ressourcenaustauschs in diesem Zeitschritt.

**Interaktionen**

Die Interaktion zwischen Agenten und Umgebung ist so gestaltet, dass sie die Prinzipien der Effectuation-Theorie, wie z.B. das „Bird in Hand“-Prinzip und das „Affordable Loss“-Prinzip, in den Entscheidungsprozessen und Attributen der Agenten widerspiegelt. Direkte Interaktionen zwischen der Umgebung und den Agenten sind in diesem Modell weitestgehend ausgeschlossen, mit der Ausnahme des Szenarios, in dem ein Agent aufgrund unzureichender Ressourcen aus dem System eliminiert wird. Eine wechselseitige Beeinflussung zwischen der Umgebung und den Agenten erfolgt jedoch in jedem einzelnen Zeitschritt des Modells.

Die Interaktionen zwischen den Agenten und ihren Entscheidungsprozessen werden durch eine Reihe von Methoden und Funktionen innerhalb des Modells gesteuert.

Beispielsweise gibt es Funktionen, die es den Agenten ermöglichen, Netzwerke zu bilden und aufzulösen, basierend auf zufälligen Auswahlprozessen oder aufgrund des Bedarfs eines Agenten hinsichtlich einer bestimmten Ressource. Dies spiegelt die dynamische Natur von Unternehmensnetzwerken wider, in denen Beziehungen ständig geknüpft und aufgelöst werden.

Eine genaue Beschreibung aller möglichen Interaktionsmethoden findet sich in Kapitel 5.2.5 wieder.

**Datenstrukturen**

Das Modell verwendet verschiedene Datenstrukturen, um die Attribute der Akteure, die verfügbaren Ressourcen und die Interaktionen zwischen den Akteuren zu speichern. Hierzu gehören Listen, Dictionaries und benutzerdefinierte Klassen in Python. Die Datenstrukturen sind so konzipiert, dass sie leicht erweiterbar sind, um zusätzliche Attribute oder Methoden aufzunehmen, die für zukünftige Forschungsfragen relevant sein könnten.

**Parametrisierung**

Ein weiterer wichtiger Aspekt der Implementierung ist die flexible Parametrisierung. Die Simulationsparameter wie die Anzahl der Agenten, die Dauer der Simulation und die initialen Ressourcen wurden so gewählt, dass sie einen realistischen Rahmen für die Anwendung der Effectuation-Prinzipien bieten. Diese Parameter können jedoch einfach angepasst werden, um verschiedene Szenarien zu simulieren und die Robustheit des Modells zu testen. Das vorgestellte Modell verfügt über mehr als 50 Parameter, die sich jederzeit anpassen lassen, um spezielle Situationen darzustellen und die Sensitivität des Modells in verschiedenen Kontexten zu testen. Diese hohe Anzahl an anpassbaren Parametern steigert die Flexibilität und Anwendbarkeit des Modells und ermöglicht auch eine tiefgreifende Analyse der Auswirkungen einzelner Variablen auf die Gesamtergebnisse, jedoch verfügt das Modell über keine Benutzeroberfläche. Die Parameter der Simulation sind im Code selbst einzustellen und die Ergebnisse sind bei Ausführung in Echtzeit beobachtbar.

**Visualisierung**

Schließlich bietet das Modell zur vorhandenen Textausgabe eine Reihe von Analyse- und Visualisierungstools, die es ermöglichen, die Ergebnisse der Simulation zu interpretieren und zu visualisieren. Diese Tools ermöglichen, die Verteilung der Ressourcen unter den Agenten zu analysieren, die Auswirkungen verschiedener Parameter auf die Ergebnisse zu untersuchen und die zukünftige Entwicklung des Modells vorherzusagen.

### 5.2.3 Spezielle Algorithmen

In diesem Kapitel wird eine detaillierte Untersuchung der Algorithmen vorgenommen, die im Modell zur Anwendung kommen. Diese Algorithmen sind nicht nur für die Funktionalität des Modells von zentraler Bedeutung, sondern auch für die Interpretierbarkeit und Validität der Simulationsergebnisse. Die Algorithmen, die in diesem Abschnitt eingehend behandelt werden, sind der tanh-Algorithmus, der $\varepsilon$-greedy-Algorithmus und der Q-Learning-Algorithmus.

#### Tanh-Algorithmus

Der tanh-Algorithmus, auch als hyperbolischer Tangens bekannt, wird zur Normalisierung von Eingabewerten eingesetzt. Dies ist insbesondere dann von Bedeutung, wenn die Eingabewerte eine hohe Varianz aufweisen und in einem standardisierten Format für die weitere Verarbeitung benötigt werden. Der tanh-Algorithmus transformiert Eingabewerte in einen Bereich zwischen $-1$ und $1$, wobei die strukturellen Eigenschaften der Daten erhalten bleiben.

#### $\varepsilon$-greedy Algorithmus

Die $\varepsilon$-greedy-Strategie ist eine einfache, aber effektive Methode für die Auswahl von Aktionen im Kontext von Reinforcement Learning. Sie dient dazu, einen Kompromiss zwischen Exploration und Exploitation zu finden:

- Exploration: Der Agent erkundet die Umgebung, um neue Informationen zu sammeln. Dabei besteht die Möglichkeit, bessere Strategien zu entdecken.
- Exploitation: Der Agent nutzt die bereits gesammelten Informationen, um die Aktion auszuwählen, die die höchste erwartete Belohnung bietet.

Die ε-greedy Strategie funktioniert folgendermaßen:

- Ein Zufallswert zwischen $0$ und $1$ wird generiert.
- Wenn der Zufallswert kleiner als $\varepsilon$ ist, führt der Agent eine zufällige Aktion aus (Exploration).

- Wenn der Zufallswert gleich oder größer als $\varepsilon$ ist, führt der Agent die Aktion mit dem höchsten geschätzten Q-Wert für den aktuellen Zustand aus (Exploitation).

*Parameter: $\varepsilon$*

- Der Parameter $\varepsilon$ steuert das Verhältnis von Exploration zu Exploitation.
- Ein hoher $\varepsilon$-Wert (z.B. $0{,}9$) bedeutet, dass der Agent häufiger zufällige Aktionen ausführt und somit die Umgebung mehr erkundet.
- Ein niedriger $\varepsilon$-Wert (z.B. $0{,}1$) bedeutet, dass der Agent hauptsächlich die Aktionen ausführt, die den höchsten geschätzten Wert haben, und somit mehr ausnutzt, was er bereits gelernt hat.

*Zeitabhängiges $\varepsilon$*

In vielen Anwendungen wird $\varepsilon$ im Laufe der Zeit reduziert, sodass der Agent anfangs mehr erkundet und später mehr ausnutzt. Dies wird als "annealing" von $\varepsilon$ bezeichnet.

*Beispiel*

Angenommen, $\varepsilon = 0{,}1$ und der Agent hat die Optionen, entweder "investieren" oder "sparen" als Aktionen in einem Finanzmarkt-Szenario zu wählen. Wenn der Zufallswert, der generiert wurde, $0{,}08$ beträgt (was kleiner als $0{,}1$ ist), würde der Agent eine zufällige Aktion auswählen. Wenn der Zufallswert jedoch $0{,}15$ beträgt (was größer oder gleich $0{,}1$ ist), würde der Agent die Aktion mit dem höchsten geschätzten Q-Wert für den aktuellen Zustand auswählen.

**Q-Learning-Algorithmus**

Der Q-Learning-Algorithmus von Watkins & Dayan (1992) ist ein Modell-freies Reinforcement-Learning-Verfahren, das es den Agenten ermöglicht, optimale Handlungsstrategien in einer gegebenen Umgebung zu erlernen. Er ist wie folgt definiert.

*Zustand ($s$)*

Ein Zustand ist eine Darstellung der aktuellen Situation oder des Kontexts, in dem sich der Agent befindet. In einem Labyrinth wäre beispielsweise die aktuelle Position des Agenten ein Zustand.

*Aktion ($a$)*

Eine Aktion ist ein Schritt, den der Agent im aktuellen Zustand ausführen kann. Im Kontext eines Labyrinths könnten die möglichen Aktionen beispielsweise "nach Norden gehen", "nach Süden gehen", "nach Osten gehen" und "nach Westen gehen" sein.

*Belohnung ($r$)*

Eine Belohnung ist ein numerischer Wert, der dem Agenten für das Ausführen einer Aktion in einem bestimmten Zustand gegeben wird. Die Belohnung dient als Feedback und repräsentiert den Nutzen der Aktion.

*Q-Wert ($Q(s,a)$)*

Der Q-Wert ist eine Schätzung des erwarteten kumulativen zukünftigen Nutzens (der zukünftigen Belohnungen), den der Agent erhalten wird, wenn er eine bestimmte Aktion a im Zustand s ausführt und danach optimal handelt. Der Q-Wert für jedes Paar aus Zustand und Aktion wird in einer Q-Tabelle gespeichert.

*Lernrate ($\alpha$)*

Die Lernrate steuert, wie stark der Agent die neuen Informationen nutzt, um die Q-Werte zu aktualisieren.

*Diskontierungsfaktor ($\gamma$)*

Der Diskontierungsfaktor bestimmt, wie stark zukünftige Belohnungen im Vergleich zu sofortigen Belohnungen gewichtet werden. Ein hoher Wert von γ bedeutet, dass der Agent langfristige Belohnungen höher bewertet.

*Q-Learning-Formel*

Die Q-Werte werden mit der folgenden Formel aktualisiert:

$$Q(s,a) = (1-\alpha) \cdot Q(s,a) + \alpha \cdot \left(r + \gamma \cdot max\left(Q(s',a')\right)\right)$$

*Ablauf*

- Der Agent beginnt in einem Anfangszustand $s$.
- Der Agent wählt eine Aktion $a$ basierend auf einer Politik (z.B. *ε*-greedy).
- Der Agent führt die Aktion $a$ aus und beobachtet die Belohnung *r* und den neuen Zustand *s'*.
- Der Q-Wert $Q(s,a)$ wird mit der oben genannten Formel aktualisiert.
- Der Agent wechselt in den neuen Zustand $s'$ und der Prozess wiederholt sich.

Der Algorithmus aktualisiert also den erwarteten Nutzen (Q-Wert) jeder Aktion in jedem Zustand basierend auf den erhaltenen Belohnungen und den zukünftigen erwarteten Belohnungen. Dies ermöglicht den Agenten, ihre Strategien im Laufe der Zeit anzupassen und zu optimieren. In dem vorliegenden Modell ist der Zustandsraum als ein Tupel konzipiert, welches den vorherigen Zustand, die ausgewählte Aktion sowie die korrespondierende Marktsituation umfasst. Die Erweiterung des Zustandsraums um zusätzliche Entscheidungseigenschaften, insbesondere in einem Q-Learning-Kontext, kann jedoch zu einem Problem führen, das als "Fluch der Dimensionalität" bekannt ist. Mit jedem zusätzlichen Merkmal, das dem Zustand hinzugefügt wird, wächst der Zustandsraum exponentiell. Dieser Ansatz könnte die Notwendigkeit erhöhen, dass der Agent eine umfangreichere Erfahrungsbasis sammeln muss, um eine adäquate Abdeckung des Zustandsraums zu erzielen und effektives Lernen zu ermöglichen. Um sowohl die Lernfähigkeit der Agenten als auch die Anwendbarkeit des erworbenen Wissens zu optimieren, wurde das Eingabetupel in diesem Modell auf die zuvor genannten drei zentralen Merkmale reduziert.

Der Q-Learning-Algorithmus und die $\varepsilon$-greedy-Strategie werden oft gemeinsam verwendet, aber sie sind nicht zwangsläufig miteinander verknüpft. Die Kombination der beiden ist besonders nützlich, weil Q-Learning alleine nicht vorschreibt, wie eine Aktion ausgewählt werden soll. Es aktualisiert lediglich die Q-Werte, die die erwarteten zukünftigen Belohnungen für verschiedene Aktionen in verschiedenen Zuständen repräsentieren. Die $\varepsilon$-greedy-Strategie bietet dann einen Mechanismus, um diese Q-Werte zur Aktionsauswahl zu verwenden, während sie gleichzeitig eine gewisse Wahrscheinlichkeit für die Exploration neuer Aktionen beibehält.

Jeder dieser Algorithmen hat spezifische Anwendungsgebiete innerhalb des Modells und trägt zu dessen Gesamtfunktionalität bei.

### 5.2.4 Ablaufbeschreibung des Modells

#### Kurzbeschreibung

Die Simulation beginnt mit der Initialisierung des Modells, bei der eine festgelegte Anzahl von Agenten erzeugt wird. Jeder Agent wird mit einer festgelegten oder zufälligen Menge an finanziellen, humanen und sozialen Ressourcen ausgestattet. Ein spezieller Agent, bekannt als Agent 100, wird jedoch mit eigens definierten und fixierten Ressourcen initialisiert. Parallel dazu wird die Marktlage als stochastische Variable zwischen $0{,}35$ und $0{,}65$ festgelegt. Nach der Initialisierung tritt das Modell in seine Hauptschleife ein. In jedem Zeitschritt ("Step") werden Daten über den aktuellen

Zustand der Agenten und des Marktes gesammelt. Die Agenten sowie der Markt führen verschiedene Aktionen aus, die wie folgt ausgearbeitet sind:

1. Zeitschritt Beginnt: Zu Beginn jedes Zeitschritts wird überprüft, ob ein unerwartetes Ereignis mit einer bestimmten Wahrscheinlichkeit eintritt. Falls ja, werden die entsprechenden Auswirkungen auf den Markt und die Agenten angewandt.
2. Marktaktualisierung: Die Marktbedingung wird für den aktuellen Zeitschritt aktualisiert. Dies beeinflusst später die Entscheidungen der Agenten sowie die Erlöse aus Geschäftsmöglichkeiten.
3. Agentenaktivierung: Die Agenten werden in zufälliger Reihenfolge aktiviert, um Geschäftsmöglichkeiten auszuführen.
    a. Entscheidungsfindung: Jeder Agent legt sich nun seine eigene Strategie zurecht und entscheidet basierend auf seinem Lernverhalten, ob er investieren oder sparen möchte.
    b. Ressourcenaustausch: Der Agent versucht einen Ressourcenaustausch und interagiert dazu mit anderen Agenten. Die Erfolgsrate dieses Austauschs hängt von verschiedenen Faktoren ab, darunter die Verlusttoleranz beider Agenten und die aktuelle Marktlage.
    c. Entscheidungsauflösung: Nach einer festgelegten Verzögerungsperiode wird die zuvor getroffene Entscheidung umgesetzt, und der Agent erhält die kalkulierte Belohnung, oder im Falle einer Misswirtschaft, Bestrafung.
4. Ressourcenprüfung: Nachdem alle Aktionen abgeschlossen sind, überprüft die Umgebung, ob es Agenten gibt, deren Ressourcen unter einen kritischen Wert gefallen sind. Ist das der Fall, scheidet der Agent aus dem Modell aus ("Ruhestand").
5. Datensammlung: Nachdem alle Agenten ihre Aktionen abgeschlossen haben, sammelt der DataCollector Daten für den aktuellen Zeitschritt. Dies umfasst sowohl agentenspezifische Variablen als auch Modellvariablen.
6. Zeitschritt Endet: Der Zeitschritt ist abgeschlossen und das Modell bereitet sich auf den nächsten Zeitschritt vor.

Dieser Zyklus wiederholt sich für die vorgegebene Anzahl von Zeitschritten. Am Ende der Simulation werden verschiedene Analysemethoden angewendet, um die gesammelten Daten zu interpretieren und zu visualisieren und um Einblicke in die Dynamik des modellierten Systems zu gewinnen. Es ist auch möglich, das Modell mehrfach

mit unterschiedlichen Parametern oder Anfangsbedingungen per eingebauter Replikationsmethode auszuführen, um Mittelwerte und Standardabweichungen aller Agenten und der Umgebung zu prüfen. Abbildung 2 zeigt nun den kompletten zeitlichen Ablauf des Modells.

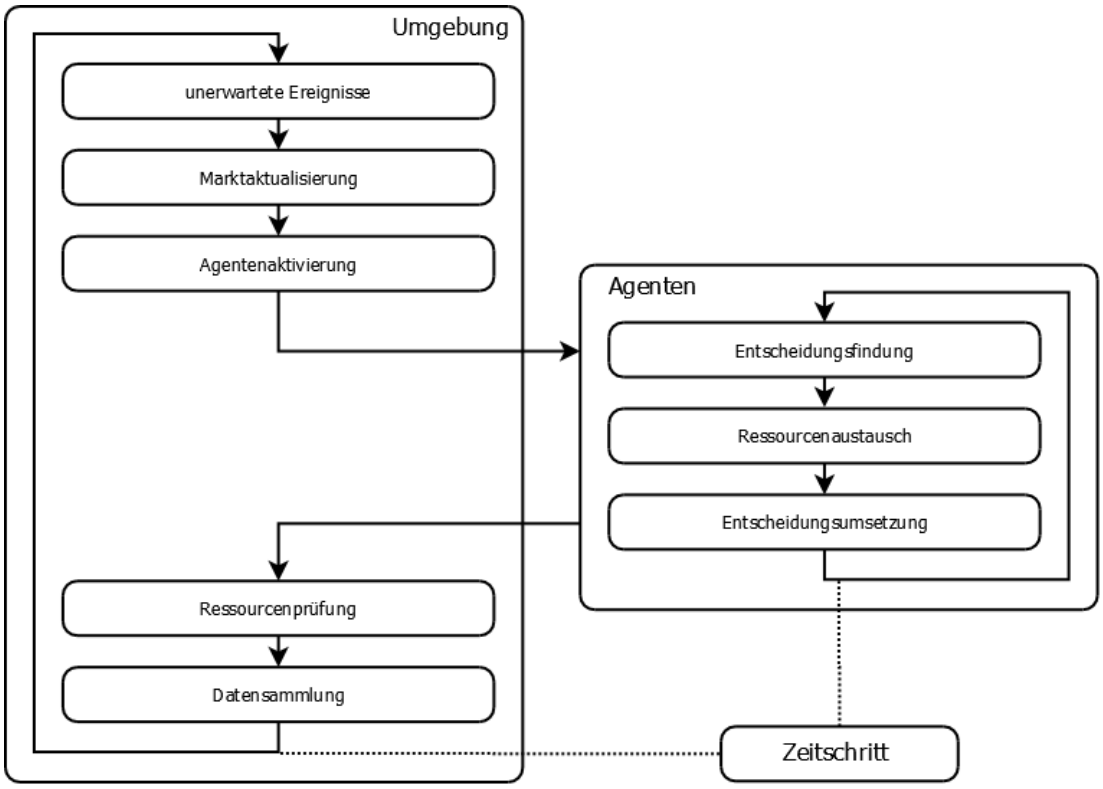


Abbildung 2: Ablauf des Simulationsmodells

### 5.2.5 Detaillierter Modellaufbau und UML-Diagramm

Für eine bessere Orientierung und ein tieferes Verständnis der Struktur des Gesamtmodells wird ein UML-Diagramm in Abbildung 3 präsentiert. Der vollständige, zugehörige Python-Programmcode des konzipierten Modells ist in Anhang A nachzulesen. Folgend werden die wichtigsten Attribute und Methoden der Klassen beschrieben.

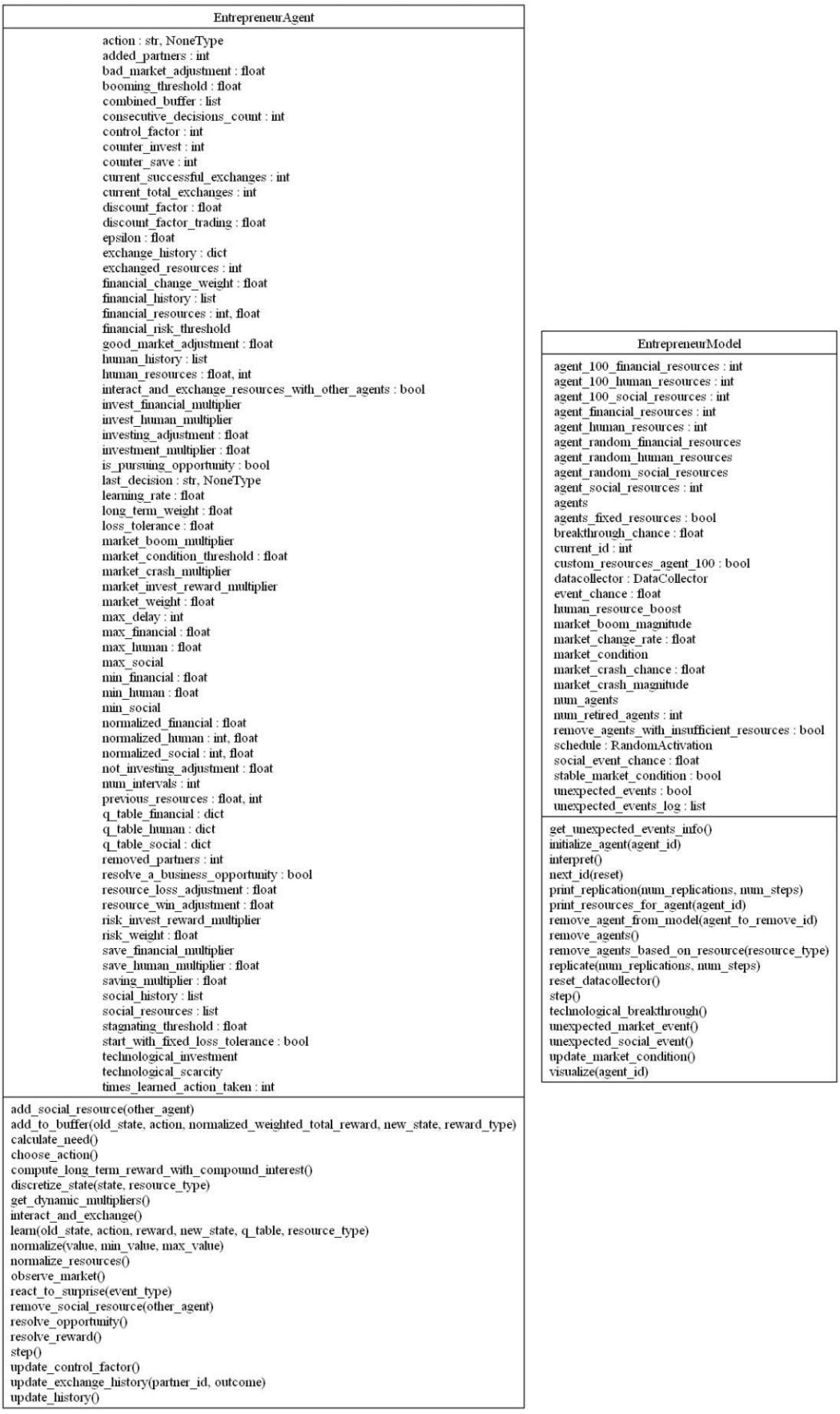


Abbildung 3: Komponenten des Simulationsmodells

**Klasse: EntrepreneurModel**

Die *EntrepreneurModel*-Klasse ist das Herzstück der Simulation und verwaltet die Agenten sowie die globale Marktsituation. Sie enthält mehrere Attribute und Methoden, die für die Durchführung der Simulation entscheidend sind. Die wichtigsten werden folgend aufgeführt.

***Attribute:***

*num_agents*: Die Anzahl der Agenten in der Simulation.

*schedule*: Ein Scheduler, der die Reihenfolge der Agentenaktionen steuert.

*market_condition*: Ein Wert zwischen 0 und 1, der den aktuellen Zustand des Marktes repräsentiert.

*unexpected_events_log*: Ein Log für unerwartete Ereignisse in der Simulation.

*num_retired_agents*: Die Anzahl der Agenten, die aus der Simulation ausgeschieden sind.

*datacollector*: Ein Objekt der DataCollector-Klasse, das zur Sammlung und Speicherung von Simulationsdaten verwendet wird.

***Methoden:***

*update_market_condition()*

ist verantwortlich für die Aktualisierung des Marktzustands unter Berücksichtigung von zwei Hauptfaktoren: dem Verhalten der Agenten und der Marktsättigung. Die Marktsättigung wird als Verhältnis der Anzahl der erfolgreichen Austauschaktionen zur Gesamtzahl aller Austauschaktionen berechnet. Wenn keine Austauschaktionen durchgeführt wurden, wird die Marktsättigung als null festgelegt. Das Verhalten der Agenten ist das Verhältnis der Anzahl der investierenden Agenten zur Gesamtanzahl aller Agenten definiert. Der neue Marktzustand wird durch die Kombination der Verhaltensmuster der Agenten und der Marktsättigung, jeweils gewichtet durch einen Faktor, der die maximale Änderung kennzeichnet, berechnet. Diese Methode wird in jedem Simulationszyklus (Zeitschritt) aufgerufen und dient als Mechanismus, um die dynamischen Wechselwirkungen zwischen den Agenten und dem Markt abzubilden.

*unexpected_market_event()*

simuliert unerwartete Marktereignisse, die entweder einen Markteinbruch (market crash) oder einen Marktaufschwung (market boom) darstellen können. Die Methode

wird in jedem Zeitschritt aufgerufen und entscheidet anhand einer Wahrscheinlichkeitsvariable, ob ein solches Ereignis eintritt. Im Falle eines Ereignisses wird der Marktzustand entsprechend angepasst und die Agenten reagieren auf das unerwartete Ereignis.

`technological_breakthrough()`

simuliert einen technologischen Durchbruch, der die menschlichen Ressourcen eines zufällig ausgewählten Agenten steigert. Es wird angenommen, dass dieser Agent eine bahnbrechende technologische Entdeckung gemacht hat und für sich verbuchen kann. Auch hier reagieren die anderen Agenten auf dieses unerwartete Ereignis.

*unexpected_social_event()*

simuliert ein unerwartetes soziales Ereignis, das die sozialen Ressourcen eines zufällig ausgewählten Agenten erhöht. Auch hier reagieren die anderen Agenten auf das unerwartete Ereignis.

`step()`

ist besonders wichtig, da sie die Hauptlogik der Simulation steuert. Sie sammelt zunächst Daten, entfernt Agenten, deren Ressourcen erschöpft sind, und führt dann unerwartete Ereignisse und Agentenaktionen aus. Schließlich aktualisiert sie den Marktzustand.

Zusätzlich gibt es mehrere Methoden, um Grafiken zu generieren und sinnvolle Simulationsausgabewerte zu berechnen.

**Klasse: EntrepreneurAgent**

Die `EntrepreneurAgent`-Klasse modelliert ein Unternehmen in einem Marktumfeld. Diese Klasse erbt von der Agent-Klasse aus dem *Mesa*-Framework und fügt spezifische Attribute und Methoden hinzu, die für die Simulation des unternehmerischen Verhaltens relevant sind.

***Attribute:***

`financial_resources`: Finanzielle Ressourcen des Agenten.

`human_resources`: Die menschlichen Ressourcen (Fähigkeiten, Talent usw.) des Agenten.

`social_resources`: Soziale Ressourcen des Agenten, repräsentiert als Liste anderer Agenten.

*loss_tolerance*: Die Verlusttoleranz des Agenten, beeinflusst Entscheidungen unter verschiedenen Marktbedingungen.

*epsilon:* Der $\varepsilon$-Wert für den $\varepsilon$-greedy-Algorithmus, der zwischen Exploration und Exploitation abwägt.

*learning_rate:* Parameter für den Q-Learning-Algorithmus.

*action:* Die zuletzt gewählte Aktion des Agenten, entweder „investieren“ oder „sparen“.

*control_factor:* Ein Faktor, der die Fähigkeit des Agenten zur Kontrolle seiner Umgebung darstellt.

*combined_buffer:* Ein Puffer für Erfahrungen, die für das spätere Lernen verwendet werden.

*discount_factor_trading:* Ein Parameter, der Ressourcen, die durch die *interact_and_exchange* Methode getauscht werden, auf- oder abwertet.

***Methoden:***

*normalize_resources()*

ist dafür verantwortlich, die Ressourcen eines Agenten zu standardisieren. In dieser Klasse gibt es drei Hauptarten von Ressourcen: finanzielle, menschliche und soziale Ressourcen. Jede dieser Ressourcen hat ihre eigene Skala und Einheit, die für die Modellierung und Analyse unpraktisch sein können.

Die Methode nimmt die aktuellen Werte dieser Ressourcen und skaliert sie auf einen Bereich zwischen $0$ und $1$. Dies geschieht, indem der aktuelle Wert der Ressource vom Minimalwert subtrahiert und dann durch die Differenz zwischen dem Maximal- und dem Minimalwert dividiert wird. Falls der Maximalwert gleich dem Minimalwert ist (was zu einer Division durch null führen würde), wird die normalisierte Ressource auf null gesetzt.

Das Normalisieren der Ressourcen hat den Vorteil, dass die Agenten ihre Entscheidungen auf einer gemeinsamen Skala treffen können. Es erleichtert auch die Anwendung von Algorithmen des maschinellen Lernens und der statistischen Analyse, da die Eingangsdaten in einer standardisierten Form vorliegen.

*calculate_need()*

ist dafür zuständig, den Bedarf eines Agenten für verschiedene Arten von Ressourcen zu ermitteln. Diese Ressourcen sind in drei Hauptkategorien unterteilt: finanzielle,

menschliche und soziale Ressourcen. Der Bedarf für jede dieser Ressourcen wird als das inverse Element des jeweiligen normalisierten Werts berechnet. Das bedeutet, wenn der Agent beispielsweise viele finanzielle Ressourcen hat, wird der finanzielle Bedarf gering sein und umgekehrt.

Nachdem der Bedarf für jede Ressource einzeln berechnet wurde, werden diese Bedürfnisse normalisiert, sodass sie zusammen eins ergeben. Das gestattet, die Bedürfnisse als eine Art Wahrscheinlichkeitsverteilung zu betrachten, die angibt, welche Ressource der Agent im aktuellen Moment am dringendsten benötigt.

Sollte der unwahrscheinliche Fall eintreten, dass die Summe aller Bedürfnisse null ist (was bedeuten würde, dass der Agent in allen Bereichen gut aufgestellt ist), werden die Bedürfnisse gleichmäßig auf alle Ressourcen verteilt, sodass jede ein Drittel der Gesamtbedürfnisse ausmacht.

Diese Methode spielt eine wichtige Rolle für die Entscheidungsfindung des Agenten. Sie liefert eine Grundlage dafür, welche Aktionen der Agent in seinem nächsten Schritt vornehmen könnte, abhängig davon, welche Ressourcen er am meisten benötigt.

*update_history()*

dient dazu, die Historie eines Agenten zu aktualisieren. Diese Historie ist eine Aufzeichnung der Zustände und Aktionen, die der Agent im Laufe der Zeit durchlaufen hat. Die Historie ist besonders für die Entscheidungsfindung des Agenten und für die Analyse seines Verhaltens über die Zeit wichtig.

In der Methode wird ein neuer Eintrag in die History-Liste des Agenten hinzugefügt. Dieser Eintrag ist ein Wörterbuch, das verschiedene Schlüssel-Wert-Paare enthält:

- *financial_resources*: Der aktuelle Stand der finanziellen Ressourcen des Agenten.
- *human_resources*: Der aktuelle Stand der menschlichen Ressourcen des Agenten.
- *social_resources*: Die Anzahl der sozialen Ressourcen, die der Agent aktuell hat.
- *action*: Die zuletzt vom Agenten ausgeführte Aktion, entweder "investieren" oder "sparen".
- *market_condition*: Der aktuelle Zustand des Marktes, der aus dem übergeordneten Modell abgeleitet ist.

- *successful_exchanges*: Die Anzahl der erfolgreichen Ressourcenaustausche, die der Agent in diesem Zeitschritt hatte.
- *total_exchanges*: Die Gesamtanzahl der Ressourcenaustausche, die der Agent in diesem Zeitschritt hatte.

Die Methode *update_history()* ist entscheidend für die Langzeitanalyse des Agentenverhaltens und für die Optimierung seiner Strategien. Durch die Aufzeichnung dieser Informationen kann der Agent aus der Vergangenheit lernen und seine zukünftigen Aktionen besser anpassen. Es ermöglicht auch eine detaillierte Analyse des Verhaltens des Agenten im Kontext des gesamten Modells.

*learn()*

verwendet den Q-Learning-Algorithmus, um die Q-Tabelle des Agenten zu aktualisieren. Die Q-Tabelle ist ein Kernbestandteil des Q-Learning-Prozesses und speichert die geschätzten Werte (Q-Werte) für die Kombinationen aus Zuständen und Aktionen. Diese Q-Werte repräsentieren den erwarteten kumulativen zukünftigen Nutzen für das Ergreifen einer bestimmten Aktion in einem bestimmten Zustand.

Hier ist eine Schritt-für-Schritt-Erklärung der Methode:

- Zustände diskretisieren: Die Methode diskretisiert die alten und neuen Zustände, um sie in der Q-Tabelle zu verwenden. Die Diskretisierung ist notwendig, um kontinuierliche Zustände in eine endliche Anzahl von diskreten Zuständen umzuwandeln, die in der Q-Tabelle gespeichert werden können.
- Marktbedingung diskretisieren: Die Marktbedingung wird ebenfalls diskretisiert, um sie in der Q-Tabelle zu verwenden.
- Alten Q-Wert abrufen: Der alte Q-Wert für die Kombination aus diskretem, altem Zustand, Aktion und diskreter Marktbedingung wird aus der Q-Tabelle abgerufen.
- Maximalen zukünftigen Q-Wert berechnen: Der maximale Q-Wert für den neuen Zustand $s'$ über alle möglichen Aktionen wird berechnet.
- Neuen Q-Wert berechnen: Der neue Q-Wert wird unter Verwendung der oben genannten Q-Learning-Formel berechnet.
- Q-Tabelle aktualisieren: Der neue Q-Wert wird in der Q-Tabelle gespeichert, sodass der Agent in zukünftigen Entscheidungen von dieser Erfahrung profitieren kann.

Die Methode gibt keinen Wert zurück, sondern aktualisiert nur die Q-Tabelle des Agenten.

*choose_action()*

ist dafür verantwortlich, die nächste Aktion für den Agenten auszuwählen. Sie stellt eine Art der Geschäftsmöglichkeit dar und verwendet einen $\varepsilon$-greedy-Algorithmus, um die Balance zwischen Exploration (neue Aktionen ausprobieren) und Exploitation (beste bekannte Aktion ausführen) zu finden. Hier sind die wichtigsten Schritte der Methode:

- Initialisierung von Epsilon: Zuerst wird der Wert von Epsilon ($\varepsilon$) initialisiert. Dies ist die Wahrscheinlichkeit, mit der der Agent eine zufällige Aktion auswählt, anstatt die Aktion mit dem höchsten erwarteten Nutzen zu nehmen.
- Diskretisierung des aktuellen Zustands: Der aktuelle Zustand des Agenten wird in diskrete Intervalle für jede Ressourcendimension (finanziell, menschlich, sozial) und den Marktstatus eingeteilt. Dies ist notwendig, da der Q-Learning-Algorithmus mit diskreten Zuständen arbeitet.
- Zufällige Aktion mit Wahrscheinlichkeit $\varepsilon$: Mit einer Wahrscheinlichkeit von $\varepsilon$ wird eine zufällige Aktion ("investieren" oder "sparen") ausgewählt.
- Aktion basierend auf Q-Werten: Wenn keine zufällige Aktion gewählt wurde, werden die Q-Werte für die möglichen Aktionen im aktuellen Zustand aus den Q-Tabellen abgerufen.
- Berechnung des durchschnittlichen Q-Werts: Der Durchschnitt der Q-Werte für jede mögliche Aktion ("investieren" oder "sparen") wird berechnet.
- Auswahl der Aktion: Die Aktion mit dem höchsten durchschnittlichen Q-Wert wird ausgewählt. Wenn die durchschnittlichen Q-Werte für beide Aktionen gleich sind, wird eine zufällige Aktion gewählt.
- Aktualisierung der action-Eigenschaft des Agenten: Schließlich wird die ausgewählte Aktion in der action-Eigenschaft des Agenten gespeichert.

Diese Methode ist entscheidend für das Verhalten des Agenten, da sie bestimmt, welche Aktion der Agent in jedem Zeitschritt ausführen wird. Sie nutzt sowohl die Historie des Agenten als auch den aktuellen Zustand des Marktes und der Ressourcen des Agenten, um eine informierte Entscheidung zu treffen.

*resolve_opportunity()*

ist zuständig für die Auflösung der Konsequenzen einer Aktion ("investieren" oder "sparen") des Agenten. Sie wird aufgerufen, nachdem der Agent eine Entscheidung getroffen hat, und aktualisiert die finanziellen, menschlichen und sozialen Ressourcen des Agenten entsprechend.

- Initialisierung von Variablen und Konstanten: Hier werden alle benötigten Werte initialisiert. Das schließt sowohl die aktuellen Zustände der Ressourcen als auch die Multiplikatoren und Gewichtungen für die verschiedenen Arten von Belohnungen ein.
- Berechnung der Konsequenzen der Entscheidung:
    - Wenn die Entscheidung "investieren" ist, werden die finanziellen und menschlichen Ressourcen entsprechend verändert. Es werden auch Markt- und Risikobelohnungen berechnet. Zudem gibt es eine kleine Chance, dass eine soziale Ressource hinzugefügt wird.
    - Wenn die Entscheidung "sparen" ist, ändern sich ebenfalls die finanziellen und menschlichen Ressourcen, allerdings auf eine andere Weise. Es gibt auch eine kleine Chance, dass eine soziale Ressource verloren geht.
- Berechnung der langfristigen Belohnungen: Die Methode prüft, ob die aktuelle Entscheidung die gleiche wie die vorherige ist, um die Anzahl der aufeinanderfolgenden gleichen Entscheidungen zu aktualisieren. Dies wird verwendet, um eine langfristige Belohnung zu berechnen.
- Gewichtung der finanziellen Belohnungen: Die finanzielle Veränderung, langfristige Belohnung und Markt- und Risikobelohnungen werden gewichtet und summiert.
- Normalisierung der Belohnungen für den Q-Learning-Algorithmus: Alle berechneten Belohnungen und Veränderungen werden mittels des tanh-Algorithmus normalisiert, um die Werte zwischen $-1$ und $1$ zu begrenzen.
- Aktualisierung des neuen Zustands: Die neuen Zustände der finanziellen, menschlichen und sozialen Ressourcen werden berechnet und gespeichert.
- Erfahrungspuffer: Der alte Zustand, die getroffene Entscheidung, die berechnete Belohnung und der neue Zustand werden im Erfahrungspuffer des Agenten für zukünftiges Lernen gespeichert.

Die Methode gibt nichts zurück und dient hauptsächlich der internen Aktualisierung des Zustands des Agenten.

*interact_and_exchange()*

steuert den Interaktions- und Austauschprozess zwischen verschiedenen Agenten in der Simulation, und stellt eine weitere Art der Geschäftsmöglichkeit dar, da sich je nach Parametrisierung mehr Ressourcen akkumulieren lassen. Sie implementiert

das, was Sarasvathy (2001) als „Crazy Quilt“-Prinzip bezeichnet, bei dem zusammengearbeitet wird, um gemeinsame Ziele zu erreichen. Hier sind die Hauptkomponenten und Schritte dieser Methode:

- Partnerauswahl: Der Agent wählt einen anderen Agenten aus, mit dem er interagieren möchte. Dies basiert auf verschiedenen Kriterien, wie z.B. dem Bedarf an bestimmten Ressourcen und der Historie der vorherigen Interaktionen. Der Agent versucht hier aktiv alle Agenten, mit denen er Partnerschaften pflegt, gleichmäßig oft zu selektieren.
- Anfrage für Ressourcenaustausch: Der Agent sendet eine Anfrage an den ausgewählten Partner, um bestimmte Ressourcen auszutauschen. Dies könnte finanzielle, menschliche oder soziale Ressourcen umfassen.
- Verhandlung und Entscheidung: Beide Agenten bewerten die Anfrage und treffen eine Entscheidung, ob sie den Austausch akzeptieren oder ablehnen. Dies hängt von verschiedenen Faktoren ab wie der aktuellen Ressourcenlage, der Historie der Interaktionen sowie der eigenen Verlusttoleranz, welche als Begrenzung des maximal ausgetauschten Ressourcenvolumens dient.
- Austauschdurchführung: Wenn beide Agenten zustimmen, wird der Austausch durchgeführt und die Ressourcen werden entsprechend aktualisiert. Hier kann es je nach Einstellung des Parameters *discount_factor_trading* zur Auf- oder Abwertung einer getauschten Ressource kommen.
- Aktualisierung der Austauschgeschichte: Unabhängig davon, ob der Austausch erfolgreich war oder nicht, wird die *exchange_history* für beide Agenten aktualisiert. Dies dient als Aufzeichnung der Interaktionen und kann für zukünftige Entscheidungen eine Rolle spielen.

Diese Methode ist wichtig für die Modellierung der sozialen Dynamik und der Ressourcenverteilung in einem Unternehmensökosystem. Sie ermöglicht den Agenten, aktiv zu interagieren und ihre Ressourcen in einer Art und Weise zu verwalten, die ihre Ziele und Strategien widerspiegelt.

*update_control_factor()*

ist dafür zuständig, den Kontrollfaktor des Agenten zu aktualisieren. Dieser Faktor ist ein Maß für das Ausmaß, in dem der Agent erfolgreich Interaktionen mit anderen Agenten hat. Konkret wird der Kontrollfaktor als das Verhältnis der Anzahl der positiven Austausche zur Gesamtzahl aller Agenten berechnet.

- Gesamtzahl der Agenten: Zunächst wird die Gesamtzahl der Agenten im Modell ermittelt.
- Initialisierung eines Zählers: Ein Zähler für die Anzahl der Agenten, mit denen dieser Agent eine netto positive Austauschhistorie hat, wird initialisiert.
- Durchlaufen der Austauschhistorie: Die Methode iteriert durch die Austauschhistorie des Agenten. Für jeden Partner wird die Nettoanzahl der Austausche berechnet. Dies ist die Differenz zwischen der Anzahl der erfolgreichen und der abgelehnten Austausche.
- Zählen der positiven Agenten: Wenn die Nettoanzahl der Austausche mit einem Partner positiv ist, wird der Zähler für positive Agenten inkrementiert.
- Berechnung des Kontrollfaktors: Schließlich wird der Kontrollfaktor als das Verhältnis der Anzahl der Agenten mit positiven Austauschen zur Gesamtzahl der Agenten berechnet. Es gibt eine spezielle Behandlung für den Fall, dass die Gesamtzahl der Agenten null ist, um eine Division durch null zu vermeiden.

Die Methode gibt keinen Wert zurück, sondern aktualisiert lediglich den internen Zustand des Agenten, insbesondere den Kontrollfaktor.

*resolve_reward()*

ist dafür zuständig, die Belohnungen oder Strafen für den Agenten basierend auf seinen aktuellen Aktionen und seinem Zustand zu bearbeiten. Dies geschieht im Kontext eines Erfahrungswiederholungspuffers, der verwendet wird, um verzögerte Belohnungen oder Strafen anzuwenden. Die Methode aktualisiert die finanziellen, menschlichen und sozialen Ressourcen des Agenten und aktualisiert auch die Q-Tabellen für das Lernen.

Hier ist eine Übersicht über die wichtigsten Schritte:

- Durchlaufen des Erfahrungspuffers: Die Methode iteriert durch den Erfahrungspuffer und betrachtet jeden Eintrag einzeln.
- Auspacken der Einträge: Jeder Eintrag im Puffer wird in seine Bestandteile zerlegt: Alter Zustand (*old_state*), Aktion (*action*), Belohnung (*reward*), neuer Zustand (*new_state*), Verzögerung (*delay*) und Belohnungstyp (*reward_type*).
- Prüfen der Verzögerung: Für jeden Eintrag wird geprüft, ob die Verzögerungszeit abgelaufen ist.

    - Belohnung oder Strafe anwenden: Wenn die Verzögerung abgelaufen ist, wird die Belohnung oder Strafe entsprechend ihres Typs angewendet:
    - Eintrag aus dem Puffer entfernen: Nach der Anwendung der Belohnung oder Strafe wird der Eintrag aus dem Puffer entfernt.
- Aktualisieren der Verzögerung: Wenn die Verzögerung noch nicht abgelaufen ist, wird der Verzögerungszähler um eins verringert und der Eintrag im Puffer entsprechend aktualisiert.

Die Methode gibt keinen Wert zurück, sondern aktualisiert nur den internen Zustand des Agenten, insbesondere seine Ressourcen und Q-Tabellen für das weitere Lernen.

*observe_market()*

beobachtet die aktuellen Marktbedingungen und passt die Verlusttoleranz des Agenten entsprechend an. Die Verlusttoleranz ist ein interner Zustandsparameter des Agenten, der seine Bereitschaft zu riskanten Investitionen widerspiegelt. Die Anpassung erfolgt basierend auf verschiedenen Faktoren, darunter die allgemeinen Marktbedingungen, die Ressourcengewinne oder -verluste des Agenten seit dem letzten Schritt und die Aktionen anderer Agenten am Markt.

Hier ist eine Übersicht über die wichtigsten Schritte der Methode:

- Definieren der Anpassungsfaktoren: Es werden vordefinierte Anpassungsfaktoren für verschiedene Bedingungen festgelegt wie z.B. für gute und schlechte Marktbedingungen oder für den Fall, dass der Agent Ressourcen gewonnen oder verloren hat.
- Berechnung der Ressourcenänderung: Die Änderung der Ressourcen des Agenten (finanziell und menschlich) seit dem letzten Schritt wird berechnet.
- Beobachtung der Marktbedingungen: Je nach positiver oder negativer Marktlage wird auch die Verlusttoleranz des Agenten geändert.
- Anpassung der Verlusttoleranz basierend auf Ressourcengewinn oder -verlust: Wenn der Agent seit dem letzten Schritt Ressourcen gewonnen hat, wird seine Verlusttoleranz vergrößert, im Falle eines Verlustes verringert.
- Anpassung der Verlusttoleranz basierend auf den Aktionen anderer Agenten: Wenn mehr als die Hälfte der Agenten am Markt investiert haben, wird die Verlusttoleranz des Agenten ebenfalls positiv angepasst, ansonsten negativ.

- Begrenzung der Verlusttoleranz: Schließlich wird die Verlusttoleranz des Agenten zwischen $0$ und $0{,}8$ begrenzt, um sicherzustellen, dass sie innerhalb eines sinnvollen Bereichs bleibt.

Die Methode gibt keinen Wert zurück, sondern aktualisiert lediglich den internen Zustand des Agenten.

### 5.2.6 Spezifische Schalterparameter

Diese speziellen Schalterparameter dienen als flexibles Instrumentarium zur Modifikation des Agenten- und Umgebungsverhaltens. Sie erlauben, bestimmte Methoden und Aktionen innerhalb der Modellinstanz selektiv zu aktivieren oder zu deaktivieren. Diese Funktionalität ist besonders nützlich, um unterschiedliche Testszenarien zu konstruieren und so die Robustheit und Sensitivität des Modells zu evaluieren, oder benutzerdefinierte Hypothesentests durchzuführen.

**EntrepreneurAgent**

*self.resolve_a_business_opportunity = True*

Wenn dieser Parameter auf "*True*" gesetzt ist, wird die Methode zur Auflösung von Geschäftsmöglichkeiten für jeden Agenten ausgeführt. Die *resolve_opportunity()* Methode kann ausgeführt werden.

*self.interact_and_exchange_resources_with_other_agents = True*

Wenn dieser Parameter auf "*True*" gesetzt ist, können die Agenten Ressourcen mit anderen Agenten austauschen. Die *interact_and_exchange()* Methode kann ausgeführt werden.

*self.start_with_fixed_loss_tolerance = True*

Wenn dieser Parameter auf "*True*" gesetzt ist, beginnen alle Agenten mit einer festgelegten Verlusttoleranz, anstatt diese zufällig zu initialisieren.

**EntrepreneurModel**

*self.unexpected_events = True*

Dieser Parameter steuert, ob im Modell unerwartete Ereignisse simuliert werden, die das Verhalten der Agenten und die Marktbedingungen beeinflussen können.

*self.stable_market_condition = False*

Wenn dieser Parameter auf "*True*" gesetzt ist, bleibt die Marktlage stabil und erlebt keine Schwankungen, was die Dynamik des Modells beeinflusst.

*self.remove_agents_with_insufficient_resources = True*

Wenn dieser Parameter auf "*True*" gesetzt ist, kann die Umgebung Agenten aus dem Modell entfernen, wenn ihre Ressourcen unter ein bestimmtes Niveau fallen, um die Realitätsnähe zu erhöhen.

*self.agents_fixed_resources = True*

Wenn dieser Parameter auf "*True*" gesetzt ist, werden alle Agenten mit festgelegten Ressourcen initialisiert, anstatt sie zufällig zu zuweisen.

*self.custom_resources_agent_100 = False*

Wenn dieser Parameter auf "*True*" gesetzt ist, wird Agent 100 mit benutzerdefinierten Ressourcenwerten initialisiert, ansonsten werden die Ressourcenwerte zufällig zugewiesen.

### 5.2.7 Integration der Effectuation-Prinzipien in das Simulationsmodell

In der Entrepreneurship-Forschung hat sich die Effectuation-Theorie als einflussreiches Paradigma etabliert, das den Entscheidungsfindungsprozess von Unternehmenden beschreibt. Die Theorie postuliert fünf Hauptprinzipien: „Bird in Hand“-Prinzip, „Affordable Loss“-Prinzip, „Lemonade“-Prinzip, „Crazy Quilt“-Prinzip und „Pilot in the Plane“-Prinzip. Im folgenden Abschnitt wird dargelegt, wie diese fünf Prinzipien in dem entwickelten agentenbasierten Modell (ABM) implementiert sind.

**„Bird in Hand“-Prinzip**

Dieses Prinzip besagt, dass Unternehmende mit dem arbeiten, was sie bereits haben: ihre Fähigkeiten, ihr Wissen und ihre Ressourcen. Im Modell wird dieses Prinzip durch die Ressourcen selbst und die *normalize_resources* Methode implementiert. Jeder Agent bewertet seine finanziellen, menschlichen und sozialen Ressourcen im Kontext der aktuellen Marktsituation. Dies bildet die Grundlage für die Wahl von Aktionen und Entscheidungen in späteren Phasen.

**„Affordable Loss“-Prinzip**

Laut diesem Prinzip setzen Unternehmende nur so viel ein, wie sie zu verlieren bereit sind, anstatt potenzielle Gewinne zu maximieren. Dies wird durch den *loss_tolerance* Parameter und die *observe_market* Methode operationalisiert, in der der Agent seine Verlusttoleranz anhand der aktuellen Marktlage stets neu bewertet.

**„Lemonade“-Prinzip**

Dieses Prinzip besagt, dass Unternehmende unvorhergesehene Umstände als Gelegenheiten nutzen, um neue Wege zu finden. Im Modell wird dies durch den Parameter *control_factor* und die *update_control_factor* Methode repräsentiert. Der Kontrollfaktor des Agenten wird basierend auf der Anzahl der positiven Austausche mit anderen Agenten aktualisiert, was dem Agenten erlaubt, unerwartete Ereignisse besser zu überstehen, oder diese sogar zu seinem eigenen Vorteil zu nutzen.

**„Crazy Quilt"-Prinzip**

Unternehmende bauen Partnerschaften und Kooperationen auf, um Risiken zu teilen und Ressourcen zu bündeln. Dieses Prinzip wird durch die Methode *interact_and_exchange* dargestellt. In dieser Methode interagiert der Agent mit anderen Agenten, um Ressourcen auszutauschen, was die eigene Handlungsfähigkeit und kollektive Widerstandsfähigkeit und Anpassungsfähigkeit erhöht.

**„Pilot in the Plane"-Prinzip**

Das letzte Prinzip besagt, dass Unternehmende die Zukunft durch ihre Handlungen gestalten, anstatt sie vorherzusagen. Dies wird im Modell durch die *interact_and_exchange* und die *choose_action* Methode verwirklicht, in der der Agent seine Strategien dynamisch an die Marktgegebenheiten und seine vergangenen Erfahrungen anpasst.

Alle anderen wichtigen Parameter und Konzepte finden sich ebenfalls explizit oder implizit in den vorhandenen Attributen und Methoden des Simulationsmodells wieder.

Insgesamt integriert das agentenbasierte Modell die Effectuation-Prinzipien auf eine Weise, die den komplexen, nicht-linearen und dynamischen Charakter des unternehmerischen Entscheidungsprozesses einfängt. Die Implementierung dieser Prinzipien ermöglicht dem Modell, realistischere Szenarien der unternehmerischen Aktivität und des Marktverhaltens zu simulieren.

### 5.2.8 Simulationsausgaben

Ein kritischer Aspekt jeder Simulationsstudie sind die Ausgaben der Simulation, die sorgfältig analysiert und interpretiert werden müssen, um fundierte Schlussfolgerungen ziehen zu können. Die Simulationsausgaben können eine Vielzahl von Datenpunkten umfassen, darunter Zustandsvariablen der Agenten, Interaktionsmuster, Marktindikatoren und weitere aggregierte Metriken. Diese Daten sind nicht nur für die Validierung des Modells selbst von Bedeutung, sondern auch für die Extrapolation der Ergebnisse auf reale Szenarien.

Um einen methodischen Rahmen für den Vergleich zu schaffen, wurde in dieser Simulation ein spezieller Agent, Agent 100, als Benchmark oder Referenzpunkt innerhalb des Systems eingeführt. Während die meisten Agenten im Modell entweder zufällig oder mit vordefinierten Anfangswerten für ihre Ressourcen initialisiert werden, wird Agent 100 gezielt mit vordefinierten Ressourcenwerten ausgestattet. Diese Konfiguration ermöglicht, Agent 100 als eine Art "Idealtypus" zu betrachten, gegen den die Performance und Entscheidungsfindung der anderen Agenten evaluiert werden können. Der zentrale Fokus liegt dabei auf der vergleichenden Analyse zwischen Agent 100 und dem durchschnittlichen Verhalten der restlichen Agenten in Bezug auf ihre Ressourcenentwicklung, ihre Entscheidungsfindung und ihre Interaktionen mit dem Markt und anderen Agenten. Durch diesen Vergleich wird ein vertieftes Verständnis der Mechanismen und Strategien angestrebt, die zu erfolgreicher oder weniger erfolgreicher unternehmerischer Tätigkeit führen.

Diese differenzierte Betrachtung gestattet nicht nur die Identifikation von Best Practices oder erfolgreichen Strategien, sondern liefert auch wertvolle Einblicke in die Wechselwirkungen und Abhängigkeiten innerhalb des agentenbasierten Systems. Dadurch kann das Modell nicht nur als Werkzeug zur wissenschaftlichen Analyse, sondern auch als Grundlage für praxisrelevante Empfehlungen dienen.

In der nachfolgenden Sektion werden exemplarische Ausgaben des entwickelten agentenbasierten Simulationsmodells präsentiert.

Für eine umfassende Analyse und Interpretation der Simulationsresultate werden sowohl textuelle als auch graphische Ausgaben generiert. Die textuellen Ausgaben dienen als quantitative Darstellung der Simulationsdaten, wobei spezifische Metriken und Indikatoren in tabellarischer Form ausgegeben werden. Diese ermöglichen eine detaillierte Untersuchung der Verhaltensmuster der Agenten sowie der Dynamik des Gesamtsystems.

Parallel dazu werden graphische Ausgaben erstellt, die eine visuelle Analyse der Simulationsdaten ermöglichen. Diese beinhalten verschiedene Arten von Diagrammen und Plots, durch die räumliche und zeitliche Trends, Beziehungen zwischen Variablen und emergente Phänomene auf intuitive Weise veranschaulicht werden. Die Kombination von textuellen und graphischen Ausgaben bietet ein umfassendes Instrumentarium für die wissenschaftliche Auswertung der Simulationsergebnisse.

**Textausgaben**

Eine Textausgabe des Simulationsmodells gibt folgende Informationen wieder.

*---------- Results of simulation ----------*

*# Investments: 5588 & # Saves: 4412*

*# Retired agents due to insufficient financial resources: 0*

*Number of unexpected events: 9*

*At step 8, unexpected event of type social event occurred.*

*At step 19, unexpected event of type social event occurred.*

…

*Exchange history of agent 100:*

*With agent 96: 2 successful exchanges, 1 refused exchanges*

*With agent 50: 2 successful exchanges, 1 refused exchanges*

…

| | Parameter Type | Agent 100 | Mean | Standard Deviation |
|---|---|---|---|---|
| **0** | Financial resources | 160.302183 | 171.063507 | 95.586750 |
| **1** | Human resources | 198.623181 | 164.124299 | 72.650391 |
| **2** | Social resources | 43.000000 | 45.620000 | 5.447532 |
| **3** | Loss Tolerance | 0.800000 | 0.763310 | 0.072761 |
| **4** | Control Factor | 0.460000 | 0.357100 | 0.089781 |

*---------- Results of replicated simulation ----------*

*Replications of model: 5*

*0 agents retired due to insufficient financial resources over the course of 5 replications*

*Number of unexpected events: 31*

*At step 2, unexpected event of type market boom occurred.*

*At step 3, unexpected event of type technological breakthrough occurred.*

…

Diese exemplarische, abgekürzte textuelle Darstellung dient als illustrative Demonstration der Modellergebnisse und ermöglicht eine initiale quantitative Analyse der simulierten Dynamiken.

Zunächst wird die Anzahl der durchgeführten Aktionen des Typs "Investieren" und "Sparen" für alle Agenten im Modell präsentiert. Dies wird gefolgt von einer Aufstellung der Agenten, die infolge unzureichender Ressourcen in den Ruhestand versetzt wurden. Anschließend wird die Quantität der aufgetretenen unerwarteten Ereignisse dargelegt, begleitet von einem detaillierten Log, welches die zeitliche Abfolge dieser unerwarteten Ereignisse dokumentiert.

Des Weiteren wird die Austauschhistorie des Agent 100 beleuchtet. Diese gibt Auskunft darüber, mit welchen anderen Agenten erfolgreiche und nicht erfolgreiche Ressourcenaustausch-Aktionen durchgeführt wurden.

Abschließend präsentiert das Modell einen Dataframe, eine spezialisierte Form einer Tabelle. Jede Zeile dieses Dataframes repräsentiert einen spezifischen Parameter und zeigt dessen Endwert für den Benchmark-Agenten Agent 100. Darüber hinaus enthält der Dataframe den Durchschnittswert des jeweiligen Parameters für alle anderen Agenten sowie die zugehörige Standardabweichung. Die erfassten Parameter umfassen finanzielle Ressourcen, humane Ressourcen, soziale Ressourcen, Verlusttoleranz und den Kontrollfaktor.

**Grafiken**

Das Simulationsmodell generiert am Ende jedes Durchlaufs zwei spezifische Grafikausgaben.

Die erste Grafik enthält drei Histogramme, die die drei Ressourcenarten – finanzielle, humane und soziale Ressourcen – abbilden, damit diese auf Normalverteilung geprüft werden können. Innerhalb dieser Histogramme wird der Balken, der Agent 100 repräsentiert, durch eine abweichende Farbgebung hervorgehoben. Aufgrund der verwendeten Visualisierungsbibliothek *matplotlib* ergibt sich eine Limitation: Der äußerste rechte Balken des Histogramms ist rechtsoffen, was dazu führt, dass Agent 100 nicht dargestellt wird, falls er sich in dieser Kategorie befindet. Abbildung 4 visualisiert ein exemplarisches Ergebnis des Simulationsmodells mit Histogrammen in grafischer Form.

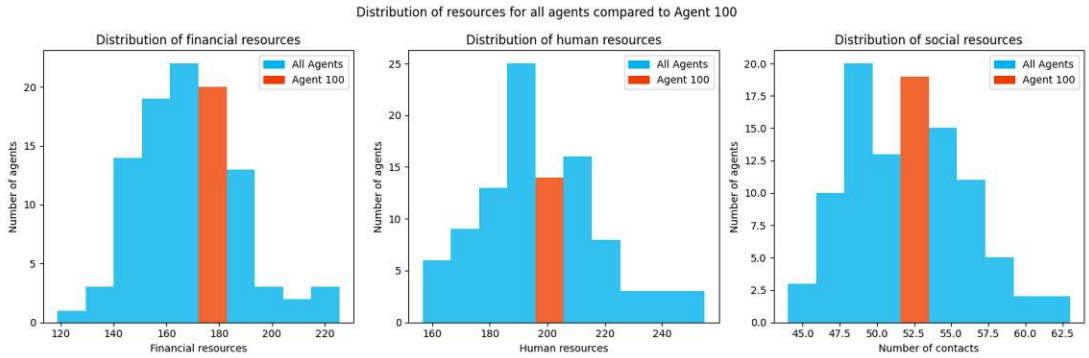


Abbildung 4 : Exemplarische Abbildung einer Simulationsausgabe I

Die zweite Grafikausgabe beinhaltet sechs Plots, die die zeitliche Entwicklung verschiedener Parameter illustrieren. In den ersten fünf Plots werden die Werte von Agent 100 in Bezug auf die durchschnittlichen Werte der übrigen Agenten für die Parameter finanzielle Ressourcen, humane Ressourcen, soziale Ressourcen, Verlusttoleranz und Kontrollfaktor dargestellt. Der sechste und letzte Plot visualisiert die jeweilige Marktlage über den Verlauf der Simulation. Abbildung 5 visualisiert ein exemplarisches Ergebnis des Simulationsmodells mit Plots in grafischer Form.

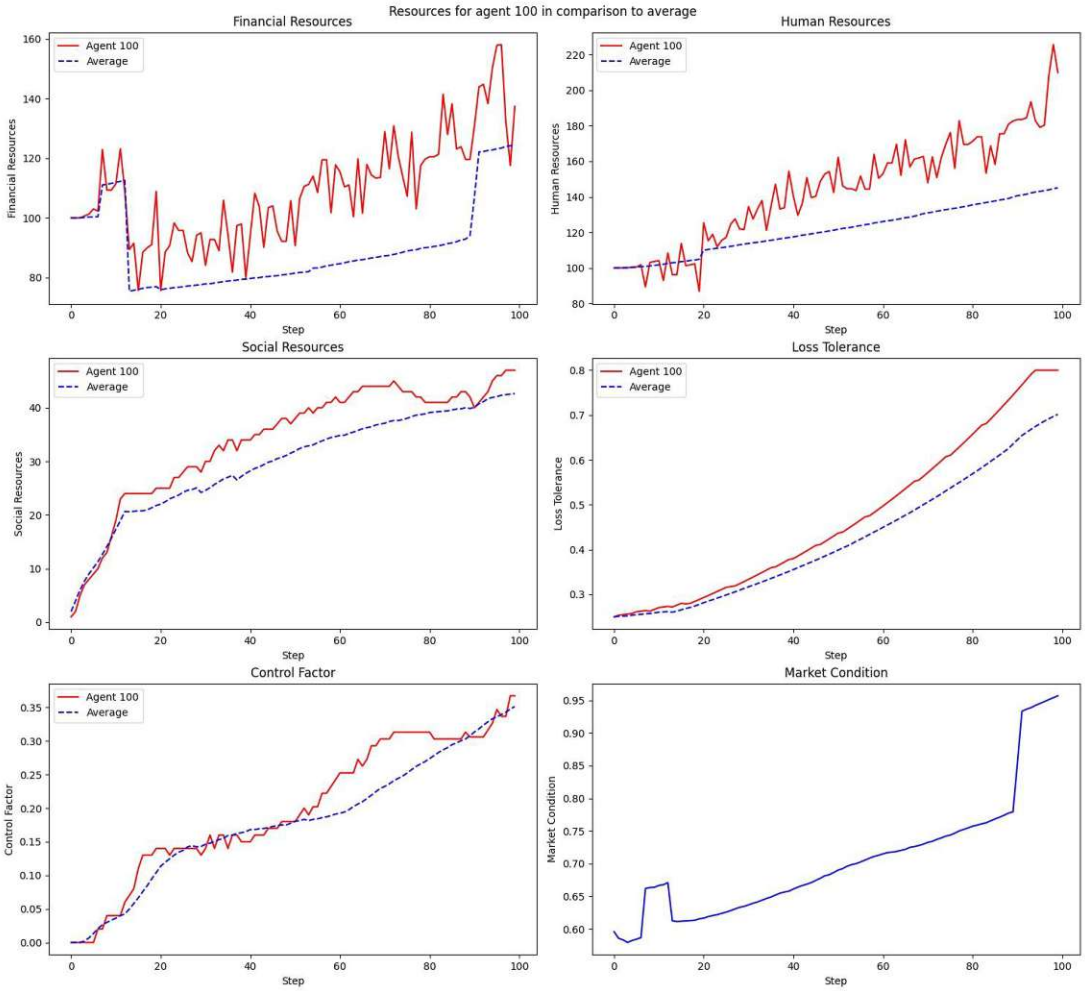


Abbildung 5: Exemplarische Visualisierung einer Simulationsausgabe II

Diese Darstellungen dienen nicht nur der Veranschaulichung, sondern bieten auch eine Basis für eine weiterführende, quantitativ-analytische Untersuchung der simulierten Systemdynamiken.

### 5.2.9 Evaluierung des Simulationsmodells

In der Evaluierungsphase des agentenbasierten Simulationsmodells wurde ein mehrschichtiges Testverfahren angewendet, um die Robustheit und Validität des Modells sicherzustellen. Spezifische Komponenten des Modells wurden gezielt deaktiviert, um isolierte Tests von individuellen Methoden und Algorithmen durchzuführen. Dies ermöglichte, die Funktionalität und Auswirkungen jeder Komponente sowohl einzeln als auch in Kombination mit anderen Elementen des Modells zu analysieren.

Um die *learn* Methode eingehend zu evaluieren, wurde ein gezieltes Testverfahren implementiert. In diesem Kontext wurde die Methode *interact_and_exchange* temporär deaktiviert, um die Einflüsse zusätzlicher Ressourcenaustauschs auf die Lernfunktionalität zu minimieren. Zudem wurde die Marktkondition auf einen stabilen Zustand festgelegt, um Schwankungen und externe Einflüsse auszuschließen. In der weiterführenden Evaluationsphase wurde der Simulationstest erneut durchgeführt, allerdings unter modifizierten Bedingungen: Der Wert für $\varepsilon$ in der $\varepsilon$-greedy-Strategie wurde auf 1 gesetzt. Dies stellt sicher, dass die Wahl zwischen den Aktionen „investieren“ und „sparen“ vollständig zufällig erfolgt. Um die Konsistenz und Reproduzierbarkeit der Ergebnisse zu gewährleisten, wurden alle randomisierten Variablen auf fixe Werte gesetzt. Durch den Vergleich der Ausgaben dieser beiden konsequent reproduzierbaren Simulationsdurchläufe konnte eindeutig belegt werden, dass der Q-Learning-Algorithmus wie erwartet funktioniert. Diese methodische Vorgehensweise gestattete eine isolierte Überprüfung der *learn* Methode und trug zur Validierung der Implementierung des zugrundeliegenden Q-Learning-Algorithmus bei.

In ähnlicher Weise wurde bei der Evaluierung der Implementierung des Kontrollfaktors verfahren. Speziell wurden die Methoden *choose_action* und *resolve_opportunity* temporär deaktiviert, um eine isolierte Untersuchung des Kontrollfaktors zu ermöglichen. Zudem wurde die Marktsituation auf einen stabilen Zustand kalibriert, um unerwünschte Schwankungen oder externe Einflüsse auszuschließen. Alle randomisierten Parameter wurden ebenfalls auf fixierte Werte gesetzt, um die Konsistenz und Reproduzierbarkeit der Testergebnisse zu gewährleisten. Diese rigorose methodologische Herangehensweise erleichterte eine präzise Analyse der Funktionsweise und Effektivität des Kontrollfaktors im Rahmen des agentenbasierten Modells.

Des Weiteren wurden Sensitivitätsanalysen durchgeführt, um die Reaktion des Modells auf Änderungen in den Eingabeparametern zu bewerten. Dies half dabei, das Verhalten des Modells unter verschiedenen Marktbedingungen und Agentenstrategien zu verstehen.

Um die Reproduzierbarkeit der Ergebnisse zu gewährleisten, wurden die Simulationen unter verschiedenen Initialbedingungen und mit unterschiedlichen Agentenkonfigurationen mehrfach ausgeführt. Jede Konfiguration wurde dabei sowohl mit als auch ohne die Einflussnahme von externen Faktoren wie unerwarteten Marktereignissen getestet.

Die Testergebnisse wurden sowohl quantitativ als auch qualitativ ausgewertet, wobei die Datenausgaben des Modells durch statistische Analysen und grafische Darstellungen ergänzt wurden. Dies gestattete eine umfassende Bewertung der Leistungsfähigkeit und Zuverlässigkeit des entwickelten agentenbasierten Simulationsmodells.

## 5.3 Limitationen der Implementierung

Obwohl das Modell eine hohe Flexibilität und Anpassungsfähigkeit bietet, gibt es bestimmte Limitationen. Dazu gehören die Abstraktion von komplexen realen Prozessen und die Annahme, dass alle Akteure rational handeln.

Eine der Hauptlimitationen in der Implementierungsphase war die Rechenzeit. Obwohl das Modell in Python unter Verwendung der *NumPy*- und *Mesa*-Bibliotheken entwickelt wurde, um eine effiziente Berechnung zu ermöglichen, kann die Simulation bei einer sehr hohen Anzahl von Agenten und Iterationen zeitintensiv sein. Dies begrenzt die Durchführbarkeit von Langzeitsimulationen oder die gleichzeitige Ausführung mehrerer Szenarien.

Ein weiterer technischer Engpass war die Speicherauslastung. Die Komplexität des Modells und die Vielzahl der Parameter erfordern einen erheblichen Speicherplatz, was bei weniger leistungsfähigen Computern zu Problemen führen kann.

Darüber hinaus stellte die Integration verschiedener Datenquellen und deren Abstraktion eine Herausforderung dar. Das Modell sollte in der Lage sein, unterschiedliche Arten von Daten zu verarbeiten, von quantitativen finanziellen Indikatoren bis hin zu qualitativen Einschätzungen der sozialen Netzwerke der Agenten. Die Harmonisierung dieser unterschiedlichen Datenformate und -strukturen war ein komplexer Prozess, der spezielle Algorithmen und Datenbereinigungsmethoden erforderte, dennoch lassen sich eventuell reale Situation nicht ausreichend korrekt abbilden.

Schließlich war die Validierung des Modells eine weitere Limitation. Aufgrund der hohen Anzahl von Parametern und der Komplexität der Interaktionen zwischen ihnen war es eine Herausforderung, die Gültigkeit des Modells umfassend zu überprüfen. Obwohl verschiedene Sensitivitätsanalysen und Tests durchgeführt wurden, bleibt die Validierung ein kontinuierlicher Prozess, der weitere Untersuchungen erfordert.

# 6 Evaluation und Ergebnisanalyse

Dieses Kapitel dient der Präsentation und kritischen Analyse der empirischen Ergebnisse, die aus der Durchführung der Simulation des agentenbasierten Modells resultierten. Hier wird nicht nur die Gültigkeit der Modellimplementierung evaluiert, sondern auch die Leistungsfähigkeit und Relevanz der Modellvorhersagen im Kontext der Effectuation-Theorie diskutiert.

Die Analyse und Interpretation der aus der Simulation gewonnenen Daten sind von zentraler Bedeutung, um die Relevanz und Anwendbarkeit der Effectuation-Theorie im Kontext der Unternehmenssimulation zu bewerten. Zur Überprüfung der Hypothese dienen nun stellvertretend drei Testszenarios, die mit dem Simulationsmodell durchlaufen und evaluiert wurden.

## 6.1 Szenario 1

### 6.1.1 Validierung von Effectuation Prinzipien: „Pilot in the Plane“-Prinzip und „Bird in Hand“-Prinzip

**Einleitung**

Im Rahmen des ersten Szenarios soll die Validität der Effectuation-Prinzipien „Pilot in the Plane“ und „Bird in Hand“ empirisch überprüft werden. Speziell wird ein Agent, im Folgenden als Agent 100 bezeichnet, mit initialen Ressourcen ausgestattet, die von den Startressourcen der übrigen Agenten im Modell abweichen. Dieses Setting ermöglicht eine systematische Analyse der Einflussfaktoren auf die Entscheidungsfindung und den Ressourcenzuwachs dieses individuellen Agenten.

**Methodologie**

*Auswahl der Variablen und Parametereinstellungen*

Für die Simulation wurden die folgenden Parametereinstellungen vorgenommen:

- Anzahl der Agenten: 100
- Anzahl der Zeitschritte: 100
- Replizierte Simulationsdurchläufe: 10

Darüber hinaus wurden spezifische Modellparameter festgelegt:

- `self.resolve_a_business_opportunity = True`: Dieser Parameter ermöglicht den Agenten, Geschäftsmöglichkeiten zu erkennen und zu nutzen.

- *self.interact_and_exchange_resources_with_other_agents = True*: Erlaubt den Agenten, Ressourcen untereinander auszutauschen.
- *self.start_with_fixed_loss_tolerance = True*: Fixiert die initiale Verlusttoleranz für alle Agenten.
- *self.unexpected_events = False*: Schließt unerwartete Ereignisse von der Simulation aus, um die Komplexität zu reduzieren.
- *self.stable_market_condition = True*: Hält die Marktlage konstant, um externe Schwankungen zu minimieren.
- *self.remove_agents_with_insufficient_resources = False*: Agenten mit unzureichenden Ressourcen werden nicht aus dem Modell entfernt.
- *self.agents_fixed_resources = True*: Die Ressourcen der Agenten werden zu Beginn der Simulation fixiert.
- *self.custom_resources_agent_100 = True*: Gestattet eine individuelle Festlegung der Startressourcen für Agent 100.

*Initiale Ressourcen*

Zum besseren Verständnis der Ausgangssituation werden die initialen Ressourcen der Agenten und von Agent 100 im Folgenden dargestellt:

*Startressourcen der Agenten*

- Finanzielle Ressourcen: 100
- Humane Ressourcen: 100
- Soziale Ressourcen: 1

*Startressourcen des Agent 100*

- Finanzielle Ressourcen = 150
- Humane Ressourcen = 150
- Soziale Ressourcen = 2

**Ausgabe des Simulationsmodells**

| Parameter Type | Agent 100 Mean | Mean |
|---|---|---|
| **Financial resources** | 258.961968 | 163.434289 |
| **Human resources** | 249.327763 | 157.937430 |

*Statistical t-tests for financial resources means*

*t-statistics: 3.451183085319153, p-value: 0.002849256817274155*

*There is a statistically significant difference between agent 100 and the average of all agents financial resources.*

*Statistical t-tests for human resources means*

*t-statistics: 3.6048733469807956, p-value: 0.0020251250174228084*

*There is a statistically significant difference between agent 100 and the average of all agents human resources.*

**Statistische Analyse**

Die statistische Analyse basierte auf einer Reihe von $t$-Tests, um die Mittelwerte der finanziellen und humanen Ressourcen zwischen Agent 100 und dem Durchschnitt aller Agenten zu vergleichen.

*Ergebnisse des $t$-Tests für finanzielle Ressourcen*

- $t$-Wert: $3{,}45$
- $p$-Wert: $0{,}0028$

*Ergebnisse des $t$-Tests für humane Ressourcen*

- $t$-Wert: $3{,}60$
- $p$-Wert: $0{,}0020$

**Interpretation und Diskussion**

Die statistische Analyse liefert signifikante Ergebnisse: Der $t$-Wert und der $p$-Wert für beide untersuchten Ressourcentypen (finanziell und human) lassen darauf schließen, dass die Unterschiede zwischen Agent 100 und den übrigen Agenten nicht zufällig sind. Diese empirische Evidenz stärkt die Annahme, dass die initiale Ressourcenausstattung von Agent 100 einen nachhaltigen Einfluss auf seine Leistung im Verlauf der Simulation hat. Dies unterstreicht die empirische Validität der im Szenario untersuchten Effectuation-Prinzipien und hat Implikationen für die Theorie und Praxis der unternehmerischen Entscheidungsfindung.

## 6.2 Szenario 2

### 6.2.1 Validierung von Effectuation Prinzipien: „Lemonade“-Prinzip

**Einleitung**

Das zweite Testszenario zielt darauf ab, die Rolle des Kontrollfaktors bei der Bewältigung unerwarteter Ereignisse in einem agentenbasierten Simulationsmodell zu erforschen. Speziell wird untersucht, wie sich der Kontrollfaktor auf die Resilienz der Agenten und deren Fähigkeit, mit unerwarteten Situationen umzugehen, auswirkt.

**Methodologie**

*Auswahl der Variablen und Parametereinstellungen*

Für eine umfassende empirische Evaluation wurden die folgenden Modellparameter festgelegt:

- Anzahl der Agenten: 100
- Anzahl der Zeitschritte: 100
- Replizierte Simulationsdurchläufe: 10

Zusätzlich wurden weitere Modellparameter für die Simulation definiert:

- `self.resolve_a_business_opportunity = True`: Erlaubt den Agenten die Identifikation und Nutzung von Geschäftsmöglichkeiten.
- `self.interact_and_exchange_resources_with_other_agents = False`: Unterbindet den Austausch von Ressourcen zwischen den Agenten.
- `self.start_with_fixed_loss_tolerance = True`: Fixiert die initiale Verlusttoleranz.
- `self.unexpected_events = True`: Aktiviert unerwartete Ereignisse.
- `self.stable_market_condition = False`: Variable Marktbedingungen.
- `self.remove_agents_with_insufficient_resources = True`: Entfernt Agenten mit unzureichenden Ressourcen.
- `self.agents_fixed_resources = True`: Fixiert die initialen Ressourcen der Agenten.
- `self.custom_resources_agent_100 = False`: Verhindert eine individuelle Parametrisierung für Agent 100.

*Experimentelle Manipulation*

Der Kontrollfaktor wird in zwei unterschiedlichen Szenarien evaluiert:

- Hohes Kontrollniveau: Der Kontrollfaktor wird auf einen hohen Wert gesetzt, um den Agenten eine höhere Resilienz gegenüber unerwarteten Ereignissen zu ermöglichen.

- Niedriges Kontrollniveau: Der Kontrollfaktor wird auf einen niedrigen Wert gesetzt, um die Agenten anfälliger für unerwartete Ereignisse zu machen.

**Ausgabe des Simulationsmodells mit erhöhtem Kontrollfaktor um 0.3**

| Parameter Type | Mean | Standard Deviation |
|---|---|---|
| **Financial resources** | 196.591559 | 23.477340 |
| **Control Factor** | 0.528970 | 0.069887 |

**Ausgabe des Simulationsmodells mit konventionellem Kontrollfaktor**

| Parameter Type | Mean | Standard Deviation |
|---|---|---|
| **Financial resources** | 142.391002 | 18.029828 |
| **Control Factor** | 0.210880 | 0.066822 |

### Statistische Analyse

Die statistische Analyse erfolgte mittels $t$-Tests, die den Einfluss des Kontrollfaktors auf die finanziellen Ressourcen der Agenten überprüfen.

- $t$-Wert: $18{,}03$
- $p$-Wert: $0{,}0000$

### Interpretation

Die statistischen Ergebnisse zeigen signifikante Unterschiede im finanziellen Wohlstand der Agenten, abhängig vom eingestellten Kontrollfaktor, unterstützt durch einen hohen $t$-Wert und einen extrem niedrigen $p$-Wert. Dies legt nahe, dass ein erhöhter Kontrollfaktor die Agenten effektiver im Umgang mit unerwarteten Ereignissen macht und somit das „Lemonade“-Prinzip der Effectuation-Theorie in der simulierten Umgebung bestätigt. Ein höherer Kontrollfaktor erlaubt den Agenten, selbst in unvorhersehbaren Situationen Chancen zu erkennen und zu nutzen. Diese Erkenntnisse erweitern unser Verständnis für die Implementierung von Effectuation-Prinzipien in agentenbasierten Modellen und bieten eine empirische Grundlage für die Weiterentwicklung von Simulationsmodellen in diesem Forschungsbereich.

## 6.3 Szenario 3

### 6.3.1 Experimentelle Steuerung: Zufällige vs. nicht-zufällige Entscheidungen

#### Einleitung

Im dritten Testszenario liegt der Fokus auf der Untersuchung der Rationalität und Effektivität der Entscheidungsfindung in einem agentenbasierten Simulationsmodell. Insbesondere wird die Wirksamkeit des Q-Learning-Algorithmus in der Entscheidungsfindung zwischen den Aktionen "investieren" und "sparen" mit einem Basisszenario kontrastiert, in dem die Agenten ihre Entscheidungen zufällig treffen.

#### Methodologie

*Auswahl der Variablen und Parametereinstellungen*

Für eine rigorose empirische Untersuchung wurden folgende Parametereinstellungen vorgenommen:

- Anzahl der Agenten: 100
- Anzahl der Zeitschritte: 100
- Replizierte Simulationsdurchläufe: 10

Zusätzlich wurden spezifische Modellparameter festgelegt:

- `self.resolve_a_business_opportunity = True`: Erlaubt den Agenten die Identifikation und Nutzung von Geschäftsmöglichkeiten.
- `self.interact_and_exchange_resources_with_other_agents = False`: Unterbindet den Austausch von Ressourcen zwischen den Agenten.
- `self.start_with_fixed_loss_tolerance = True`: Fixiert die initiale Verlusttoleranz.
- `self.unexpected_events = False`: Schließt unerwartete Ereignisse aus.
- `self.stable_market_condition = True`: Stabilisiert die Marktbedingungen.
- `self.remove_agents_with_insufficient_resources = True`: Entfernt Agenten mit unzureichenden Ressourcen.
- `self.agents_fixed_resources = True`: Fixiert die initialen Ressourcen der Agenten.
- `self.custom_resources_agent_100 = False`: Verhindert eine individuelle Parametrisierung für Agent 100.

*Experimentelle Manipulation*

Die Steuerung der Entscheidungsfindung wird durch die Manipulation des $\varepsilon$-Parameters in der $\varepsilon$-greedy-Strategie erreicht. In einem Simulationslauf wird $\varepsilon$ auf $0$ gesetzt (Exploitation-Modus) und in einem weiteren auf $1$ (Explorations-Modus), um eine vollständig zufällige Aktionsauswahl zu ermöglichen.

**Ausgabe des Simulationsmodells mit $\varepsilon = 0$**

| Parameter Type | Mean | Standard Deviation |
|---|---|---|
| **Financial resources** | 189.743732 | 34.227706 |

**Ausgabe des Simulationsmodells mit $\varepsilon = 1$**

| Parameter Type | Mean | Standard Deviation |
|---|---|---|
| **Financial resources** | 180.872847 | 27.130451 |

**Statistische Analyse**

Die statistische Auswertung erfolgte durch $t$-Tests, die die Mittelwerte und Standardabweichungen der finanziellen Ressourcen in beiden Szenarien verglichen. Die Ergebnisse zeigten einen signifikanten Unterschied, charakterisiert durch:

- $t$-Wert: $2{,}03$
- $p$-Wert: $0{,}0436$

**Interpretation**

Die statistische Analyse zeigt signifikante Unterschiede zwischen den beiden Entscheidungsstrategien. Der $t$-Wert und der $p$-Wert stützen die Hypothese, dass der Einsatz des Q-Learning-Algorithmus zu einer signifikanten Verbesserung der finanziellen Metriken der Agenten führt. Dies legt nahe, dass Q-Learning eine effektivere und rationalere Entscheidungsfindungsstrategie darstellt als ein zufälliger Ansatz. Diese Erkenntnis hat weitreichende Implikationen für das Verständnis der Entscheidungsmechanismen in agentenbasierten Modellen und bietet eine empirische Grundlage für die Weiterentwicklung und Anwendung von Lern-Algorithmen in der Simulationsforschung.

## 6.4 Limitationen der Modellimplementierung

Trotz der sorgfältigen Planung und Durchführung der Forschung gibt es einige Limitationen, die bei der Interpretation der Ergebnisse berücksichtigt werden müssen.

### 6.4.1 Forschungsansatz und Methoden

Der gewählte Forschungsansatz und die Methoden haben ihre eigenen Einschränkungen. Da die Forschung hauptsächlich auf einer computergestützten Simulation basiert, sind die Ergebnisse stark von den in das Modell eingegebenen Annahmen und Parametern abhängig. Dies könnte die Generalisierbarkeit der Ergebnisse einschränken.

### 6.4.2 Datenqualität und -zuverlässigkeit

Obwohl die Datenerhebung sorgfältig durchgeführt wurde, gibt es immer noch die Möglichkeit von Ungenauigkeiten oder Verzerrungen. Die Qualität der Daten kann durch verschiedene Faktoren wie die Auswahl der Datenquellen oder die Methoden der Datenerhebung beeinflusst werden. Darüber hinaus führt die relative Jugendlichkeit der Effectuation-Theorie dazu, dass noch nicht genügend empirische Daten vorhanden sind, um Simulationen mit hoher Genauigkeit zu kalibrieren.

### 6.4.3 Modellbeschränkungen

Das Simulationsmodell hat seine eigenen Grenzen. Es kann nicht alle möglichen Szenarien oder Variablen abdecken, die in der realen Welt existieren. Insbesondere die Abstraktion der Ressourcen und Akteure könnte die Anwendbarkeit des Modells in der Praxis einschränken.

### 6.4.4 Evaluierungsbeschränkungen

Es ist wichtig zu betonen, dass die Implementierung des agentenbasierten Simulationsmodells, obwohl es umfangreiche und vielfältige Szenarien abdeckt, in seiner Testbarkeit begrenzt ist. Aufgrund der hohen Anzahl an parametrischen Kombinationen und möglichen Szenarien ist es in der Praxis nicht möglich, jede Facette des Modells umfassend zu evaluieren. Die Komplexität des Modells, das eine Vielzahl von Parametern und Initialbedingungen enthält, begrenzt die Durchführbarkeit einer vollständigen Validierung. Daher konzentriert sich die vorliegende Arbeit auf die Untersuchung von Schlüsselszenarien und -methoden, die als repräsentativ für die wesent-

lichen Funktionalitäten und Eigenschaften des Modells angesehen werden. Diese Limitation stellt eine Herausforderung für die Generalisierbarkeit der gewonnenen Erkenntnisse dar und eröffnet Räume für zukünftige Forschungsarbeiten.

### 6.4.5 Abstraktionsniveau

Alle Attribute im Modell, einschließlich der Ressourcen, sind abstrahiert. Dies ist sowohl eine Stärke als auch eine Schwäche, da es die Modellierung vereinfacht, aber auch die Komplexität der realen Welt nicht vollständig erfassen kann.

### 6.4.6 Vergleich mit bestehender Literatur

Die Vergleichbarkeit der Simulationsergebnisse mit der bestehenden Literatur ist ebenfalls begrenzt, insbesondere wenn es um neuartige oder weniger erforschte Aspekte der Effectuation-Theorie geht.

# 7 Schlussbetrachtungen

Das abschließende Kapitel der Arbeit dient dazu, die gesamten Forschungsergebnisse zusammenzufassen, ihre Bedeutung zu diskutieren und einen Ausblick auf mögliche zukünftige Forschungsrichtungen zu geben. Es bietet auch eine Gelegenheit, die Arbeit in ihrer Gesamtheit kritisch zu reflektieren.

## 7.1 Beantwortung der Forschungsfrage

In Anbetracht der Forschungsfrage und der zugehörigen Hypothese kann festgestellt werden, dass ein computergestütztes Simulationsmodell erfolgreich entwickelt wurde, welches alle Effectuation-Prinzipien und -prozesse im Bereich der Unternehmenssimulation abbildet. Allerdings hat die kritische Diskussion der Daten deutlich gemacht, dass die Implementierung des Modells sowohl Möglichkeiten als auch Grenzen aufweist, insbesondere in Bezug auf Validität, Zuverlässigkeit und Generalisierbarkeit.

Die Ergebnisse der Simulation können dazu beitragen, die Unternehmenssimulation im Kontext der Effectuation-Theorie effektiver und effizienter zu gestalten. Die modulare Architektur ermöglicht beispielsweise eine einfache Erweiterung und Anpassung des Modells, um spezifische Forschungsfragen zu adressieren. Allerdings sind die praktischen Anwendungen derzeit noch begrenzt, da die Parameter und Daten des Modells abstrahiert sind und eine direkte Übertragung auf die reale Welt schwierig ist.

In Bezug auf die Effectuation-Theorie liefert die Simulation neue Perspektiven auf die Dynamik und die Mechanismen, die den Entscheidungsprozessen von Unternehmen zugrunde liegen.

Zusammenfassend kann gesagt werden, dass das entwickelte Simulationsmodell einen wertvollen Beitrag zur Beantwortung der Forschungsfrage leistet, indem es die Implementierbarkeit eines solchen Modells demonstriert und gleichzeitig seine Möglichkeiten und Grenzen aufzeigt. Die Ergebnisse legen eine weitere Forschung und Entwicklung in diesem Bereich nahe, um die Limitationen des aktuellen Modells zu adressieren und seinen Anwendungsbereich sowohl in der Theorie als auch in der Praxis zu erweitern.

## 7.2 Kritische Diskussion der Ergebnisse

In der folgenden kritischen Diskussion der Ergebnisse sollen die signifikanten Schwächen und Einschränkungen des entwickelten agentenbasierten Simulationsmodells im Kontext der Effectuation-Theorie umfassend erörtert werden.

**Vergleich mit bestehender Literatur**

*Eigenständige Abbildung der Effectuation-Prinzipien*: Das agentenbasierte Simulationsmodell präsentiert eine eigenständige Herangehensweise zur Abbildung der Effectuation-Prinzipien. Dies könnte als Fortschritt in der Modellierung dieser Theorie betrachtet werden, da es spezifische Nuancen und Aspekte abdeckt, die in der bisherigen Literatur möglicherweise vernachlässigt wurden.

*Datenextrapolation*: Aufgrund der spärlichen Verfügbarkeit von belastbaren Daten mussten viele Daten extrapoliert werden. Dies schränkt die Verlässlichkeit der Ergebnisse ein und eröffnet die Frage, wie gut das Modell tatsächlich die realen Prozesse widerspiegelt.

*Resilienz des Modells*: Die Robustheit des Modells ist direkt abhängig von der Qualität der zugrunde liegenden Daten. Ein Modell kann nur so präzise sein, wie die Daten, die es füttern und in diesem Fall besteht ein Mangel an umfassenden, empirischen Daten.

*Interpretationsspielraum der Effectuation-Theorie*: Die Umsetzung der Effectuation-Theorie im Modell lässt einen gewissen Interpretationsspielraum. Dies bedeutet, dass verschiedene Implementierungen der Theorie zu unterschiedlichen Ergebnissen führen könnten, selbst wenn sie auf der gleichen theoretischen Grundlage basieren.

**Diskussion der Validität, Zuverlässigkeit und Generalisierbarkeit**

*Eingeschränkte Validität*: Das entwickelte Modell kann derzeit nur einzelne Konzepte testen. Eine umfassende Validierung oder Kalibrierung ist erst dann sinnvoll, wenn eine bessere Datenbasis zur Verfügung steht. Bis dahin bleiben Fragen zur Generalisierbarkeit der Ergebnisse offen.

**Beiträge und Einschränkungen der Arbeit**

*Theoretische Beiträge*: Obwohl das Modell eine Grundlage für die Exploration der Effectuation-Theorie bietet, hat es noch keinen direkten theoretischen Beitrag geleistet. Eine detaillierte Parametrisierung und Weiterforschung könnten jedoch zu wertvollen Erkenntnissen führen.

*Modulare Architektur*: Das Modell ist modular und gut dokumentiert, was die Möglichkeit für zukünftige Erweiterungen und Anpassungen bietet.

*Praktische Anwendbarkeit*: Derzeit ist das Modell nicht für praktische Anwendungen geeignet. Die abstrakte Natur der Parameter und die eingeschränkte Darstellung des Marktes limitieren die Übertragbarkeit auf reale Szenarien.

*Innovationsgrad*: Die modulare Architektur und der Kontrollfaktor, der das „Lemonade"-Prinzip abbildet, können als innovative Aspekte des Modells betrachtet werden.

*Subjektive Interpretation*: Die Interpretation der Effectuation-Theorie ist subjektiv und könnte von anderen Forschern anders umgesetzt werden. Dies könnte als Schwäche betrachtet werden.

*Begrenzte Testmöglichkeiten*: Aufgrund des Umfangs dieser Arbeit konnten nicht alle denkbaren Szenarien und Parameterkombinationen getestet werden. Dies stellt eine signifikante Limitierung dar.

*Parameterkomplexität*: Die Vielzahl der Parameter ermöglicht eine flexible Modellierung, erschwert jedoch die Interpretation und Validierung der Ergebnisse. Modelle mit weniger Parametern könnten in diesem Kontext einfacher zu handhaben sein.

Das entwickelte agentenbasierte Simulationsmodell stellt einen eigenständigen Ansatz zur Modellierung der Effectuation-Prinzipien dar. Obwohl es in einigen Aspekten als innovativ angesehen werden kann, insbesondere in Bezug auf seine modulare Architektur und den Kontrollfaktor, weist es dennoch signifikante Einschränkungen in Bezug auf Validität, Zuverlässigkeit und Generalisierbarkeit auf. Die Qualität der Ergebnisse ist stark von der zugrunde liegenden Datenbasis abhängig, die aufgrund der spärlichen empirischen Daten oft auf Extrapolationen beruhen muss. Dies wirft ernsthafte Fragen bezüglich der Robustheit und der empirischen Überprüfbarkeit des Modells auf.

Außerdem bleibt ein erheblicher Interpretationsspielraum bei der Umsetzung der Effectuation-Theorie, was zu variablen Ergebnissen führen kann. Die praktische Anwendbarkeit des Modells ist ebenfalls begrenzt, insbesondere durch die abstrakte Parametrisierung und die eingeschränkte Darstellung des Marktes.

Die Vielzahl an Parametern bietet einerseits Flexibilität, erhöht jedoch andererseits die Komplexität und erschwert die Interpretation und Validierung. Insgesamt erfordert das Modell weitere empirische Kalibrierung und theoretische Grundlegung, um seine Eignung als Forschungsinstrument zu bestätigen.

## 7.3 Ausblick und zukünftige Forschungsrichtungen

Nach der kritischen Diskussion der Arbeit und der Bewertung ihrer Leistungen und Limitationen ist es wichtig, einen Ausblick auf zukünftige Forschungsrichtungen und mögliche Anwendungen zu geben.

**Theoretische Fragen**

*Untersuchung weiterer Aspekte der Effectuation-Theorie*: Die vorliegende Arbeit hat zwar einen substantiellen Beitrag zur Modellierung von Effectuation-Prozessen geleistet, jedoch bleibt Raum für weitere Explorationen. Insbesondere wäre es interessant, die Parametrisierung des Modells zu verfeinern, um präzisere Ergebnisse zu erzielen. Dazu könnten die Arten von Ressourcen und die Strategieausrichtung angepasst werden. Solche Modifikationen könnten zu einer besseren Abbildung der realen unternehmerischen Entscheidungsfindung beitragen.

*Abbildung von Geschäftsmöglichkeiten*: Eine weitere wichtige theoretische Fragestellung betrifft die genaue Implementierung von Geschäftsmöglichkeiten im Modell. Beispielsweise bleibt unklar, ob Agenten in der Lage sein sollten, sowohl zu investieren als auch zu sparen und gleichzeitig Ressourcen mit Partnern auszutauschen. Eine gründliche Untersuchung dieser Aspekte könnte zu einem tieferen Verständnis der Dynamiken unternehmerischen Handelns führen.

**Praktische Implementierung**

*Realitätsnähe des Modells*: Aktuell ist das entwickelte Modell zu abstrakt und realitätsfern, um unmittelbare praktische Anwendungen zu rechtfertigen. Zukünftige Forschungen könnten sich darauf konzentrieren, empirische Daten zur Kalibrierung und Validierung des Modells zu verwenden. Dies würde die Relevanz des Modells für praktische Anwendungen erhöhen und könnte zur Verbesserung von Unternehmenssimulationen im Allgemeinen beitragen.

**Tools-Entwicklung**

*Entwicklung einer grafischen Benutzeroberfläche (GUI)*: Eine intuitive GUI würde die Zugänglichkeit des Modells erheblich erhöhen und könnte dazu beitragen, von einer breiteren Benutzergruppe genutzt zu werden.

*Integration von Sensitivitätsanalysen und statistischen Tests*: Zukünftige Versionen des Modells könnten von der Einbeziehung fortgeschrittener statistischer Tests profitieren. Diese würden eine tiefere Analyse der Ergebnisse ermöglichen und könnten

zur Identifizierung von Schlüsselfaktoren für den Erfolg oder Misserfolg unternehmerischer Entscheidungen beitragen.

Insgesamt bieten diese vorgeschlagenen Forschungsrichtungen sowohl theoretische als auch praktische Wege zum Ausbau des entwickelten Simulationsmodells und zur Beantwortung offener Fragen im Bereich der Effectuation-Theorie. Sie könnten darüber hinaus die Grundlage für ein breiteres Verständnis der Mechanismen und Prinzipien der unternehmerischen Entscheidungsfindung bieten.

# Literaturverzeichnis

Arend, R. J., Sarooghi, H., & Burkemper, A. (2015). Effectuation as ineffectual? Applying the 3E theory-assessment framework to a proposed new theory of entrepreneurship. Academy of Management Review, 40(4), 630-651.

Axtell, R. L. (2000). Why agents? On the varied motivations for agent computing in the social sciences. Center on Social and Economic Dynamics Working Papers, (17).

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences, 99(Supplement 3), 7280–7287.

Chandler, G. N., DeTienne, D. R., McKelvie, A., & Mumford, T. V. (2011). Causation and effectuation processes: A validation study. Journal of Business Venturing, 26(3), 375-390.

Davis, J. P., Eisenhardt, K. M., & Bingham, C. B. (2007). Developing theory through simulation methods. Academy of Management Review, 32(2), 480-499.

Dew, N., Read, S., Sarasvathy, S. D., & Wiltbank, R. (2009). Effectual versus predictive logics in entrepreneurial decision-making: Differences between experts and novices. Journal of Business Venturing, 24(4), 287-309.

Edmonds, B., & Moss, S. (2005). From KISS to KIDS–an 'anti-simplistic' modelling approach. Multi-agent and multi-agent-based simulation, 15-30.

Epstein, J. M., & Axtell, R. (1996). Growing artificial societies: Social science from the bottom up. MIT Press.

Eubank, S., Guclu, H., Kumar, V. S., Marathe, M. V., Srinivasan, A., Toroczkai, Z., & Wang, N. (2004). Modelling disease outbreaks in realistic urban social networks. Nature, 429(6988), 180-184.

Field, A. (2013). Discovering statistics using IBM SPSS statistics. Sage.

Gilbert, N. (2008). Agent-based models. SAGE Publications.

Gilbert, N., & Troitzsch, K. G. (2005). Simulation for the social scientist. Open University Press.

Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S. K., Huse, G., Huth, A., Jepsen, J. U., Jørgensen, C., Mooij, W. M., Müller, B., Pe'er, G., Piou, C., Railsback, S. F., Robbins, A. M., ...

Wiegand, T. (2006). A standard protocol for describing individual-based and agent-based models. Ecological Modelling, 198(1–2), 115–126.

Grimm, V., Berger, U., DeAngelis, D. L., Polhill, J. G., Giske, J., & Railsback, S. F. (2010). The ODD protocol: A review and first update. Ecological Modelling, 221(23), 2760–2768.

Grimm, V., & Railsback, S. F. (2005). Individual-based modeling and ecology. Princeton University Press.

Knight, F. H. (1921). Risk, Uncertainty, and Profit. Houghton Mifflin.

Macal, C. M., & North, M. J. (2010). Tutorial on agent-based modeling and simulation. Journal of Simulation, 4(3), 151–162.

Macy, M. W., & Willer, R. (2002). From factors to actors: Computational sociology and agent-based modeling. Annual review of sociology, 28(1), 143-166.

Mauer R, Wuebker R, Schlüter J, Brettel M. Prediction and control: An agent-based simulation of search processes in the entrepreneurial problem space. Strategic Entrepreneurship Journal. 2018;12:237–260. https://doi.org/10.1002/sej.1271

Miller, J. H., & Page, S. E. (2007). Complex adaptive systems: An introduction to computational models of social life. Princeton University Press.

Mintzberg, H. (1994). The Rise and Fall of Strategic Planning. Free Press.

Perry, J. T., Chandler, G. N., & Markova, G. (2012). Entrepreneurial Effectuation: A Review and Suggestions for Future Research. Entrepreneurship Theory and Practice, 36(4), 837-861.

Popper, K. (1959). *The Logic of Scientific Discovery*. Hutchinson & Co.

Railsback, S. F., & Grimm, V. (2019). Agent-based and individual-based modeling: A practical introduction. Princeton University Press.

Read, S., Dew, N., Sarasvathy, S. D., Song, M., & Wiltbank, R. (2009). Marketing Under Uncertainty: The Logic of an Effectual Approach. Journal of Marketing, 73(3), 1-18.

Read, S., Sarasvathy, S. D., Dew, N., & Wiltbank, R. (2016). Effectual Entrepreneurship. Routledge.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., & Tarantola, S. (2008). Global sensitivity analysis: The primer. John Wiley & Sons.

Sarasvathy, S. D. (2001). Causation and Effectuation: Toward a Theoretical Shift from Economic Inevitability to Entrepreneurial Contingency. Academy of Management Review, 26(2), 243-263.

Sarasvathy, S. D. (2008). Effectuation: Elements of Entrepreneurial Expertise. Edward Elgar Publishing.

Sarasvathy, S. D., & Dew, N. (2005). New market creation through transformation. Journal of Evolutionary Economics, 15(5), 533-565.

Sargent, R. G. (2013). Verification and validation of simulation models. Journal of Simulation, 7(1), 12-24.

Sterman, J. D. (2000). Business dynamics: Systems thinking and modeling for a complex world. Irwin/McGraw-Hill.

Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. Handbook of computational economics, 2, 831-880.

Thiele, J. C., Kurth, W., & Grimm, V. (2014). Facilitating parameter estimation and sensitivity analysis of agent-based models: A cookbook using NetLogo and R. Journal of Artificial Societies and Social Simulation, 17(3), 11.

Watkins, C.J.C.H. & Dayan, P (1992).Q-learning. Machine Learning, 8, 279-292.

Welter, C., & Kim, S. (2017). Effectuation under risk and uncertainty: A simulation model. Journal of Business Venturing. 33. 10.1016/j.jbusvent.2017.11.005.

Wiltbank, R., Dew, N., Read, S., & Sarasvathy, S. D. (2006). What to Do Next? The Case for Non-Predictive Strategy. Strategic Management Journal, 27(10), 981-998.

Windrum, P., Fagiolo, G., & Moneta, A. (2007). Empirical validation of agent-based models: Alternatives and prospects. Journal of Artificial Societies and Social Simulation, 10(2), 8.

Yin, R. K. (2009). *Case Study Research: Design and Methods* (4th ed.). Sage Publications.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Anhänge

## Anhang A - Modellimplementierung in Python

```
from collections import defaultdict
from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.datacollection import DataCollector
import random
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from scipy import stats
import numpy as np
import pandas as pd

np.random.seed(0)
random.seed(0)

class EntrepreneurAgent(Agent):
    """
    An agent representing an entrepreneur in a simulated market environment.

    Attributes:
    -----------
    unique_id : int
        A unique identifier for the agent.
    financial_resources : float
        The financial resources available to the agent.
    human_resources : float
        The human resources available to the agent.
    social_resources : list
        The social resources available to the agent.
    loss_tolerance : float
        The agent's tolerance for resource loss.
    control_factor : float
        The agent's control over unexpected events.
    """

    def __init__(self, unique_id, model):
        """
```

```
Initialize a new agent.

Parameters:
-----------
unique_id : int
    A unique identifier for the agent.
model : EntrepreneurModel
    The model instance in which the agent exists.
"""

super().__init__(unique_id, model)
# Variables used for calculation of model
self.financial_resources = 0.0
self.human_resources = 0.0
self.social_resources = []
self.is_pursuing_opportunity = False
self.q_table_financial = {}
self.q_table_human = {}
self.q_table_social = {}
self.action = None
self.counter_save = 0
self.counter_invest = 0
self.combined_buffer = []
self.financial_history = []
self.human_history = []
self.social_history = []
self.max_financial = 0.0
self.min_financial = 0.0
self.max_human = 0.0
self.min_human = 0.0
self.normalized_financial = 0.0
self.normalized_human = 0.0
self.normalized_social = 0.0
self.consecutive_decisions_count = 0
self.last_decision = None
self.added_partners = 0
self.removed_partners = 0
self.exchange_history = {}
self.exchanged_resources = 0
```

```
        self.times_learned_action_taken = 0
        self.previous_resources = 0
        self.current_successful_exchanges = 0
        self.current_total_exchanges = 0
        self.control_factor = 0

        # Activate or deactivate certain conditions for testing purposes
        self.resolve_a_business_opportunity = True
        self.interact_and_exchange_resources_with_other_agents = True
        self.start_with_fixed_loss_tolerance = True

        # Parameters - change values to change behavior of agents
        self.loss_tolerance = 0.307 if self.start_with_fixed_loss_tolerance else random.uniform(0.15,
0.45) # Initialize loss tolerance
        self.cooperation_readiness = 1 # Readiness to cooperate with other agents

        # Parameters - choose action
        self.epsilon = 1 # Choose action through learning: 0; choose action randomly: 1; default: 0.1

        # Parameters - learn
        self.learning_rate = 0.1 # Learning rate for Q-Learning-Algorithm
        self.discount_factor = 0.99 # Discount factor for learning purposes

        # Parameters - discretize state
        self.num_intervals = 10 # Intervals for q-table-states, increase this to make better decisions
based on learning, but apply gained knowledge less often

        # Parameters - interact and exchange
        self.discount_factor_trading = 1 # Discount factor for trading resources; default: 1; if > 1
resource upgrading;  if < 1 resource downgrading

        # Parameters - add to buffer
        self.max_delay = 5  # After a maximum of how many steps the reward and learning effects start

        # Parameters - observe market
        self.good_market_adjustment = 1.005
        self.bad_market_adjustment = 0.995
        self.investing_adjustment = 1.005
        self.not_investing_adjustment = 0.995
        self.resource_win_adjustment = 1.005
        self.resource_loss_adjustment = 0.995
```

```
        # Parameters - react to surprise
        self.market_crash_multiplier = random.uniform(0.7,0.9)
        self.market_boom_multiplier = random.uniform(1.05,1.15)
        self.technological_investment = random.uniform(0.05,0.15)
        self.technological_scarcity = random.uniform(0.03,0.06)

        # Parameters - resolve opportunity
        self.invest_financial_multiplier = random.uniform(0.005,0.01)
        self.invest_human_multiplier = random.uniform(0.005,0.01)
        self.save_financial_multiplier = random.uniform(0.0,0.01)
        self.save_human_multiplier = 0.005
        self.market_invest_reward_multiplier = random.uniform(0.0,0.01)
        self.risk_invest_reward_multiplier = random.uniform(0,0.01)
        self.financial_risk_threshold = random.uniform(0.1,0.3)
        self.market_condition_threshold = 0.5
        self.financial_change_weight = 0.4
        self.long_term_weight = 0.3
        self.market_weight = 0.15
        self.risk_weight = 0.15

        # Parameters - get dynamic multipliers
        self.booming_threshold = 0.8
        self.stagnating_threshold = 0.5
        self.investment_multiplier = 0.2
        self.saving_multiplier = 0.1

    def update_history(self):
        """
        Update the agent's history based on recent experiences and actions.

        This method appends the agent's current financial, human, and social resources
        to their respective historical lists. This allows for tracking the agent's
        resource changes over time.

        Returns:
        --------
        None
        """
```

```
        self.financial_history.append(self.financial_resources)
        self.human_history.append(self.human_resources)
        self.social_history.append(len(self.social_resources))

    def add_to_buffer(self, old_state, action, normalized_weighted_total_reward, new_state,
reward_type):
        """
        Add an experience tuple to the agent's experience replay buffer.

        Parameters:
        -----------
        old_state : object
            The previous state of the agent before taking the action.
        action : object
            The action taken by the agent.
        normalized_weighted_total_reward : float
            The normalized and weighted total reward received after taking the action.
        new_state : object
            The new state of the agent after taking the action.
        reward_type : object
            The type of reward received (can be used for categorization or special handling).

        Returns:
        --------
        None
        """
        # Generate a random delay within the agent's maximum delay limit
        delay = random.randint(1, self.max_delay)
        # Append the experience tuple to the agent's combined buffer
        self.combined_buffer.append((old_state, action, normalized_weighted_total_reward, new_state,
delay, reward_type))

    def add_social_resource(self, other_agent=None):
        """
        Add a social resource (another agent) to the agent's social network.

        This method adds another agent to the current agent's social resources list.
        If no agent is specified, a random agent is chosen from the pool of potential
        partners. The relationship is reciprocal, meaning both agents add each other
        to their respective social resources lists.
```

```
        Parameters:
        -----------
        other_agent : EntrepreneurAgent, optional
            The agent to be added to the social network. If None, a random agent will be chosen.

        Returns:
        --------
        None
        """
        # If no agent is specified, choose a random agent from the pool of potential partners
        if random.random() > self.cooperation_readiness:
            return
        if other_agent is None:
            potential_partners = [agent for agent in self.model.schedule.agents if agent != self and
agent not in self.social_resources]
            if not potential_partners:
                return # No potential partners available
            other_agent = self.random.choice(potential_partners)

        # Add the current agent to the other agent's list of social resources
        other_agent.social_resources.append(self)
        # Add the other agent to the current agent's list of social resources
        self.social_resources.append(other_agent)
        self.added_partners += 1  # Increment the counter for added partners

    def remove_social_resource(self, other_agent=None):
        """
        Remove a social resource (another agent) from the agent's social network.

        This method removes another agent from the current agent's social resources list.
        If no agent is specified, a random agent is chosen from the pool of existing
        social resources. The relationship removal is reciprocal, meaning both agents
        remove each other from their respective social resources lists.

        Parameters:
        -----------
        other_agent : EntrepreneurAgent, optional
            The agent to be removed from the social network. If None, a random agent will be chosen.
```

```
        Returns:
        --------
        None
        """
        # If no specific agent is provided, select a random agent from the existing social resources
        if other_agent is None:
            potential_ex_partners = [agent for agent in self.model.schedule.agents if agent != self and
agent in self.social_resources]
            if not potential_ex_partners:
                return
            other_agent = self.random.choice(potential_ex_partners)

        # Remove self from the other agent's social resources if present
        if self in other_agent.social_resources:
            other_agent.social_resources.remove(self)

        # Remove the other agent from my social resources if present
        if other_agent in self.social_resources:
            self.social_resources.remove(other_agent)

        # Update the count of removed partners
        self.removed_partners += 1

    def observe_market(self):
        """
        Observe the current market conditions and update the agent's internal state variables.

        This method observes the current market conditions and adjusts the agent's loss tolerance
        based on several factors. These factors include the overall market condition, the agent's
        resource gains or losses since the last step, and the actions of other agents in the market.

        The method uses predefined adjustment factors to modify the agent's loss tolerance, which
        is then capped between 0 and 0.8.

        Returns:
        --------
        None
        """
        # Define the adjustment factors for different conditions
        good_market_adjustment = self.good_market_adjustment
```

```
        bad_market_adjustment = self.bad_market_adjustment
        investing_adjustment = self.investing_adjustment
        not_investing_adjustment = self.not_investing_adjustment
        resource_win_adjustment = self.resource_win_adjustment
        resource_loss_adjustment = self.resource_loss_adjustment

        # Calculate resource change since last step
        resource_win = (self.financial_resources + self.human_resources) - self.previous_resources
        self.previous_resources = self.financial_resources + self.human_resources

        # Observe the market condition and adjust loss tolerance accordingly
        if self.model.market_condition > 0.5:
            self.loss_tolerance *= good_market_adjustment
        else:
            self.loss_tolerance *= bad_market_adjustment

        # Adjust loss tolerance based on resource gain or loss since last step
        if resource_win > 0:
            self.loss_tolerance *= resource_win_adjustment
        else:
            self.loss_tolerance *= resource_loss_adjustment

        # Adjust loss tolerance based on actions of other agents in the market
        num_agents_investing = sum(agent.action == 'invest' for agent in self.model.schedule.agents)
        if num_agents_investing > self.model.num_agents * 0.5:
            self.loss_tolerance *= investing_adjustment
        else:
            self.loss_tolerance *= not_investing_adjustment

        # Cap loss tolerance between 0 and 0.8
        self.loss_tolerance = max(0, min(0.8, self.loss_tolerance))

    def normalize_resources(self):
        """
        Normalize the agent's financial, human, and social resources.

        This method updates the agent's resources to a normalized scale based on historical data.
        The normalization is done for financial, human, and social resources separately.
```

```
        For financial and human resources, the method uses the maximum and minimum values from the
agent's history.

        For social resources, the method calculates the ratio of current social resources to possible
new partners.



        After normalization, the agent's resources are updated with their normalized values.


        Returns:

        --------

        None

        """

        # Normalize financial resources based on historical data

        self.max_financial = max(self.financial_history) if self.financial_history else
self.financial_resources

        self.min_financial = min(self.financial_history) if self.financial_history else
self.financial_resources



        # Normalize human resources based on historical data

        self.max_human = max(self.human_history) if self.human_history else self.human_resources

        self.min_human = min(self.human_history) if self.human_history else self.human_resources



        # Normalize social resources based on historical data

        self.max_social = max(self.social_history) if self.social_history else
len(self.social_resources)

        self.min_social = min(self.social_history) if self.social_history else
len(self.social_resources)



        all_agent_ids = set(agent.unique_id for agent in self.model.schedule.agents)

        social_resources_ids = set(agent.unique_id for agent in self.social_resources)

        possible_new_partners = all_agent_ids - social_resources_ids - {self.unique_id}  # Exclude self


        if len(possible_new_partners) == 0:

            self.normalized_social = 1  # Set to 0 if there are no possible new partners

        elif len(self.social_resources) == 0:

            self.normalized_social = 0

        else:

            self.normalized_social =  len(self.social_resources) / len(possible_new_partners)



        # Update the agent's normalized resources

        self.normalized_financial = self.normalize(self.financial_resources, self.min_financial,
self.max_financial)

        self.normalized_human = self.normalize(self.human_resources, self.min_human, self.max_human)

        self.normalized_social = max(0, min(1, self.normalized_social))
```

```
def normalize(self, value, min_value, max_value):
    """
    Normalize a given value between a specified minimum and maximum value.

    If max_value is equal to min_value, the function returns 1.

    Parameters:
    -----------
    value : float
        The value to be normalized.

    min_value : float
        The minimum value in the range for normalization.

    max_value : float
        The maximum value in the range for normalization.

    Returns:
    --------
    float
        The normalized value.
    """
    # Check if the maximum and minimum values are different to avoid division by zero
    if max_value != min_value:
        # Normalize the value based on the given minimum and maximum values
        return (value - min_value) / (max_value - min_value)
    else:
        # If max_value is equal to min_value, return 1 as the normalized value
        return 1

def react_to_surprise(self, event_type):
    """
    React to a surprising event by updating the agent's state based on the type of event.

    The agent's financial, human, and social resources may be affected by the event.
    The agent's reaction is influenced by various multipliers and control factors.

    Parameters:
```

```
        ----------
        event_type : str
            The type of surprising event. Can be one of the following:
            - "market_crash": A sudden downturn in the market.
            - "market_boom": A sudden upturn in the market.
            - "technological_breakthrough": A technological advancement that affects the market.
            - "social_event": An event that affects the agent's social network.

        Returns:
        --------
        None
        """
        # Define multipliers and control factors for different types of events
        market_crash_multiplier = self.market_crash_multiplier
        market_boom_multiplier = self.market_boom_multiplier
        technological_investment = self.technological_investment
        technological_scarcity = self.technological_scarcity

        # React to a market crash by reducing financial resources
        if event_type == "market_crash":
            self.financial_resources *= min(1, market_crash_multiplier* (self.control_factor +
market_crash_multiplier))

        # React to a market boom by increasing financial resources
        elif event_type == "market_boom":
            self.financial_resources *= (self.control_factor * (market_boom_multiplier - 1) +
market_boom_multiplier)

        # React to a technological breakthrough
        elif event_type == "technological_breakthrough":
            if self.loss_tolerance > technological_investment:
                # Reduce financial resources but increase human resources
                self.financial_resources = self.financial_resources * min(1, 1 -
technological_investment + self.control_factor)
                self.human_resources = self.human_resources * (1 + technological_scarcity)

        # React to a social event
        elif event_type == "social_event":
            # Add a social resource with a probability based on the control factor
            if random.random() < self.control_factor:
                self.add_social_resource()
```

```
            else:
                # Remove a random social resource
                self.remove_social_resource()

    def choose_action(self):
        """
        Choose an action for the agent to take based on an ε-greedy algorithm.

        The agent's action is determined by considering the Q-values associated
        with the current state in each resource dimension (financial, human, social).
        The method updates the agent's `action` attribute to reflect the chosen action.

        The ε-greedy algorithm is used to balance exploration and exploitation:
        - With probability ε, a random action is chosen ('invest' or 'save').
        - With probability 1-ε, the action with the highest average Q-value is chosen.

        If the average Q-values for 'invest' and 'save' are equal, a random action is chosen.

        Returns:
        --------
        None
        """
        # Initialize epsilon for the ε-greedy algorithm
        epsilon = self.epsilon

        # Discretize the current state of the agent in each resource dimension
        current_state_financial = self.discretize_state(self.financial_resources, "financial")
        current_state_human = self.discretize_state(self.human_resources, "human")
        current_state_social = self.discretize_state(len(self.social_resources), "social")
        current_state_market_condition = 0 if self.model.market_condition < 0.5 else 1

        # With probability ε, choose a random action ('invest' or 'save')
        if random.uniform(0, 1) < epsilon:
            self.action = random.choice(['invest', 'save'])
            return

        # Retrieve Q-values from the Q-table for the current state in each resource dimension
        q_invest_financial = self.q_table_financial.get((current_state_financial, 'invest',
current_state_market_condition), 0)
```

```
        q_save_financial = self.q_table_financial.get((current_state_financial, 'save',
current_state_market_condition), 0)


        q_invest_human = self.q_table_human.get((current_state_human, 'invest',
current_state_market_condition), 0)

        q_save_human = self.q_table_human.get((current_state_human, 'save',
current_state_market_condition), 0)


        q_invest_social = self.q_table_social.get((current_state_social, 'invest',
current_state_market_condition), 0)

        q_save_social = self.q_table_social.get((current_state_social, 'save',
current_state_market_condition), 0)


        # Calculate the average Q-value for each action ('invest' and 'save')

        q_invest_avg = (q_invest_financial + q_invest_human + q_invest_social) / 3

        q_save_avg = (q_save_financial + q_save_human + q_save_social) / 3


        # Choose the action with the highest average Q-value

        if q_invest_avg > q_save_avg:

            self.action = 'invest'

        elif q_save_avg > q_invest_avg:

            self.action = 'save'

        else:

            # If the average Q-values are equal, choose a random action

            self.action = random.choice(['invest', 'save'])


    def calculate_need(self):

        """

        Calculate the agent's need for each type of resource (financial, human, social)

        based on their normalized levels.


        The need for each resource is calculated as the inverse of the normalized level

        of that resource. The needs are then normalized again to sum to 1, providing

        a distribution of needs across the three resource types.


        Returns:

        --------

        tuple(float, float, float)

            A tuple containing the normalized needs for financial, human, and social resources,

            respectively. The values will sum to 1.


            If the total need for all resources is zero, the method returns (1/3, 1/3, 1/3).
```

```
        """
        # Calculate the need for each resource type based on its normalized level
        # The need is assumed to be inversely proportional to the current level of the resource
        financial_need = 1 - self.normalized_financial
        human_need = 1 - self.normalized_human
        social_need = 1 - self.normalized_social

        # Calculate the total need across all resource types
        total_need = financial_need + human_need + social_need

        # Normalize the needs so that they sum to 1
        if total_need == 0:
            # If the total need is zero, return an equal distribution for all resources
            return 1/3, 1/3, 1/3
        else:
            # Otherwise, normalize the needs
            return financial_need / total_need, human_need / total_need, social_need / total_need

    def learn(self, old_state, action, reward, new_state, q_table, resource_type):
        """
        Update the agent's Q-table based on the reward received for taking an action in a given state.

        The method uses the Q-learning algorithm to update the value of taking a particular action in a particular state.
        The Q-value is updated based on the formula:

        Q(s, a) = (1 - α) * Q(s, a) + α * (r + γ * max(Q(s', a')))

        where:
        - Q(s, a) is the old value
        - α is the learning rate
        - r is the reward
        - γ is the discount factor
        - max(Q(s', a')) is the maximum Q-value for next state across all possible actions

        Parameters:
        -----------
        old_state : float
            The old state of the agent before taking the action.
```

```
        action : str
            The action taken by the agent. Can be either 'invest' or 'save'.

        reward : float
            The reward received for taking the action.

        new_state : float
            The new state of the agent after taking the action.

        q_table : dict
            The Q-table to be updated.

        resource_type : str
            The type of resource for which the Q-table is being updated. Can be 'financial', 'human',
or 'social'.

        Returns:
        --------
        None
        """
        # Discretize the old and new states for Q-learning
        discrete_old_state = self.discretize_state(old_state, resource_type)
        discrete_new_state = self.discretize_state(new_state, resource_type)

        # Discretize the market condition for Q-learning
        discrete_market_condition = 0 if self.model.market_condition < 0.5 else 1

        # Retrieve the old Q-value from the Q-table
        old_value = q_table.get((discrete_old_state, action, discrete_market_condition), 0.0)

        # Calculate the maximum future Q-value for the new state across all possible actions
        future_value = max([q_table.get((discrete_new_state, a), 0.0) for a in ['invest', 'save']])

        # Update the Q-value using the Q-learning formula
        new_value = old_value + self.learning_rate * (reward + self.discount_factor * future_value -
old_value)

        # Update the Q-table with the new Q-value
        q_table[(discrete_old_state, action, discrete_market_condition)] = new_value
```

```
    def discretize_state(self, state, resource_type):
        """
        Discretize the agent's continuous state into discrete intervals for Q-learning.

        The method takes a continuous state value and converts it into a discrete value based on pre-
defined intervals.
        This is useful for Q-learning, which often requires state spaces to be discrete.

        Parameters:
        -----------
        state : float
            The continuous state value that needs to be discretized.

        resource_type : str
            The type of resource for which the state is being discretized. Can be 'financial', 'human',
or 'social'.

        Returns:
        --------
        int
            An integer representing the discretized state, which will be used in the Q-table.
        """
        # Determine the maximum and minimum values for the given resource type
        if resource_type == "financial":
            max_value = self.max_financial
            min_value = self.min_financial
        elif resource_type == "human":
            max_value = self.max_human
            min_value = self.min_human
        elif resource_type == "social":
            max_value = self.max_social
            min_value = self.min_social

        # Calculate the length of each interval for discretization
        n_intervals = self.num_intervals
        interval_length = (max_value - min_value) / n_intervals

        # If all values are equal, return 0 as the discrete state
        if interval_length == 0:
            return 0
```

```
        # Calculate the discrete state based on the interval length
        discrete_state = int((state - min_value) // interval_length)

        # Ensure the discrete state is within the range [0, n_intervals - 1]
        return min(discrete_state, n_intervals - 1)

    def update_exchange_history(self, partner_id, outcome):
        """
        Update the history of exchanges with another agent based on the outcome of the interaction.

        This method updates the exchange history for both the agent calling the method and the partner agent involved
        in the exchange. The history keeps track of successful and refused exchanges.

        Parameters:
        -----------
        partner_id : int
            The unique identifier for the agent with whom the exchange took place.

        outcome : str
            The outcome of the exchange, either 'successful' or 'refused'.

        Returns:
        --------
        None
        """
        # Update the exchange history for the current agent (self)
        # If the partner_id is not already in the exchange history, initialize it with zeros for 'successful' and 'refused'
        if partner_id not in self.exchange_history:
            self.exchange_history[partner_id] = {'successful': 0, 'refused': 0}

        # Increment the count for the given outcome ('successful' or 'refused') with this partner
        self.exchange_history[partner_id][outcome] += 1

        # Find the partner agent in the model's schedule to update their exchange history
        # Use a generator expression to find the first agent with a matching unique_id, or None if not found
        partner_agent = next((agent for agent in self.model.schedule.agents if agent.unique_id == partner_id), None)
```

```
        # If the partner agent is not found, print a warning and return
        if partner_agent is None:
            print(f"Warning: No agent found with unique_id {partner_id}, outcome: {outcome}")
            return

        # Update the partner agent's exchange history
        # If the current agent's unique_id is not already in the partner's exchange history, initialize
it
        if self.unique_id not in partner_agent.exchange_history:
            partner_agent.exchange_history[self.unique_id] = {'successful': 0, 'refused': 0}

        # Increment the count for the given outcome ('successful' or 'refused') with the current agent
        partner_agent.exchange_history[self.unique_id][outcome] += 1

    def interact_and_exchange(self):
        """
        Interact and exchange resources with another agent based on the current market condition and
individual needs.

        This method identifies a suitable partner for exchange based on various criteria such as least
number of
        previous exchanges, maximum resources, and current needs. It then performs the resource
exchange and updates
        the exchange history for both agents involved.

        Note: The method assumes that the agent's social network and resource needs have been
previously calculated.

        Returns:
        --------
        None
        """
        # Define exchange rates based on market conditions
        good_market_exchange_rate = random.uniform(0.1,0.2)  # This value represents the proportion of
resources to exchange when the market condition is good
        market_exchange_rate = random.uniform(0.05,0.1)  # This value represents the proportion of
resources to exchange when the market condition is bad
        if self.model.market_condition > 0.5:
            market_exchange_rate = good_market_exchange_rate

        # Initialize variables for tracking needs and exchange outcomes
        my_financial_need, my_human_need, my_social_need = 0, 0, 0
```

```
        partner_financial_need, partner_human_need, partner_social_need = 0, 0, 0
        exchange_amount = 0.0
        partner = None
        successful_exchange = False

        # Initialize counters for successful and total exchanges
        self.current_successful_exchanges = 0
        self.current_total_exchanges = 0

        # Get sets of all agents and social resources
        all_agent_ids = set(agent.unique_id for agent in self.model.schedule.agents)
        social_resources_ids = set(agent.unique_id for agent in self.social_resources)
        possible_new_partners = all_agent_ids - social_resources_ids - {self.unique_id}  # Exclude self

        # Identify least-exchanged partners
        min_total_exchanges = min((stats.get('successful', 0) + stats.get('refused', 0) for stats in
self.exchange_history.values()), default=0)
        least_exchanged_partners = [partner_id for partner_id, stats in self.exchange_history.items()
if (stats.get('successful', 0) + stats.get('refused', 0)) == min_total_exchanges]
        never_exchanged_partners = [agent.unique_id for agent in self.social_resources if
agent.unique_id not in self.exchange_history]

        # Calculate the agent's needs
        my_financial_need, my_human_need, my_social_need = self.calculate_need()

        # Resource exchange logic for financial resources
        if max(my_financial_need, my_human_need, my_social_need) == my_financial_need and
social_resources_ids:
            # Choose a partner based on various criteria
            if never_exchanged_partners:
                partner = max((self.model.agents[partner_id] for partner_id in
never_exchanged_partners), key=lambda x: x.financial_resources, default=None)
            elif least_exchanged_partners:
                partner = max((self.model.agents[partner_id] for partner_id in
least_exchanged_partners), key=lambda x: x.financial_resources, default=None)
            else:
                partner = max((self.model.agents[partner_id] for partner_id in social_resources_ids),
key=lambda x: x.financial_resources, default=None)

            if partner == None:
                print(f"no Partner for agent {self.unique_id}")

            # Calculate partner's needs
```

```
            partner_financial_need, partner_human_need, partner_social_need = partner.calculate_need()


            # Perform the exchange if conditions are met
            if my_financial_need > partner_financial_need:
                if partner.loss_tolerance > market_exchange_rate and self.loss_tolerance > 
market_exchange_rate:
                    exchange_amount =  min(partner.financial_resources, self.financial_resources) * 
market_exchange_rate


                if exchange_amount != 0.0:
                    partner.financial_resources -= exchange_amount
                    self.human_resources -= exchange_amount
                    self.financial_resources += exchange_amount * self.discount_factor_trading
                    partner.human_resources += exchange_amount * self.discount_factor_trading
                    successful_exchange = True


            # Update exchange history
            self.update_exchange_history(partner.unique_id, 'successful' if successful_exchange else 
'refused')
            if successful_exchange:
                self.current_successful_exchanges += 1
            self.current_total_exchanges += 1


        # Resource exchange logic for human resources
        elif max(my_financial_need, my_human_need, my_social_need) == my_human_need and 
social_resources_ids:
            # Choose a partner based on various criteria
            if never_exchanged_partners:
                partner = max((self.model.agents[partner_id] for partner_id in 
never_exchanged_partners), key=lambda x: x.human_resources, default=None)
            elif least_exchanged_partners:
                partner = max((self.model.agents[partner_id] for partner_id in 
least_exchanged_partners), key=lambda x: x.human_resources, default=None)
            else:
                partner = max((self.model.agents[partner_id] for partner_id in social_resources_ids), 
key=lambda x: x.human_resources, default=None)


            if partner == None:
                print(f"no Partner for agent {self.unique_id}")


            # Calculate partner's needs
            partner_financial_need, partner_human_need, partner_social_need = partner.calculate_need()
```

```
            # Perform the exchange if conditions are met
            if my_human_need > partner_human_need:
                if partner.loss_tolerance > market_exchange_rate and self.loss_tolerance > 
market_exchange_rate:
                    exchange_amount = min(partner.human_resources, self.human_resources) * 
market_exchange_rate

                if exchange_amount != 0.0:
                    partner.human_resources -= exchange_amount
                    self.financial_resources -= exchange_amount
                    self.human_resources += exchange_amount * self.discount_factor_trading
                    partner.financial_resources += exchange_amount * self.discount_factor_trading
                    successful_exchange = True

            # Update exchange history
            self.update_exchange_history(partner.unique_id, 'successful' if successful_exchange else 
'refused')
            if successful_exchange:
                self.current_successful_exchanges += 1
            self.current_total_exchanges += 1

        # If no suitable partner for resource exchange or social need is higher, try to add a new 
social resource
        elif possible_new_partners:
            self.add_social_resource()

    def resolve_opportunity(self):
        """
        Resolve the opportunity for the agent based on the current market conditions and the action 
taken.

        This method calculates the financial, human, and social rewards or penalties based on the 
agent's action
        ('invest' or 'save'). It also considers various factors such as market conditions, risk 
tolerance, and
        long-term rewards. The method then updates the agent's state and adds the new state to the 
experience
        replay buffer for future learning.

        Returns:
        --------
        None
        """
```

```
        # Initialize variables and constants
        market_condition_threshold = 0.5
        market_reward = 0
        risk_reward = 0
        human_change = 0
        social_reward = 0
        new_state_human = 0
        invest_financial_multiplier = self.invest_financial_multiplier
        invest_human_multiplier = self.invest_human_multiplier
        save_financial_multiplier = self.save_financial_multiplier
        save_human_multiplier = self.save_human_multiplier
        market_invest_reward_multiplier = self.market_invest_reward_multiplier
        risk_invest_reward_multiplier = self.risk_invest_reward_multiplier
        financial_risk_threshold = self.financial_risk_threshold

        # Weights for each resource in the reward calculation
        financial_change_weight = self.financial_change_weight
        long_term_weight = self.long_term_weight
        market_weight = self.market_weight
        risk_weight = self.risk_weight

        # Initialize variables and constants
        old_state_financial = self.financial_resources
        old_state_human = self.human_resources
        old_state_social = len(self.social_resources)

        # Calculate the consequences of the action taken by the agent
        if self.action == "invest":
            # Financial and human resource changes when investing
            financial_change = self.financial_resources * invest_financial_multiplier
            human_change = self.human_resources * invest_human_multiplier

            # Market and risk rewards when investing
            market_reward = self.financial_resources * market_invest_reward_multiplier if
self.model.market_condition > market_condition_threshold else -self.financial_resources *
market_invest_reward_multiplier
            risk_reward = self.financial_resources * risk_invest_reward_multiplier if
self.loss_tolerance > financial_risk_threshold else -self.financial_resources *
risk_invest_reward_multiplier

            # Chance to add a social resource when investing
```

```
            if random.random() < 0.1:
                if len(self.social_resources) < len(self.model.schedule.agents) - 1:
                    social_reward += 1
            self.counter_invest += 1

        else:  # action == "save"
            # Financial and human resource changes when saving
            financial_change = self.financial_resources * save_financial_multiplier
            human_change = self.human_resources * save_human_multiplier

            # Chance to lose a social resource when saving
            if random.random() < 0.1:
                if self.social_resources:
                    social_reward -= 1
            self.counter_save += 1

        # Calculate long-term rewards based on consecutive decisions
        if self.action == self.last_decision:
            self.consecutive_decisions_count += 1
        else:
            self.consecutive_decisions_count = 1
        self.last_decision = self.action
        long_term_reward = self.financial_resources *
self.compute_long_term_reward_with_compound_interest()

        # Weight the financial rewards based on various factors
        if self.financial_resources < 0:
            weighted_financial_change = 0
            weighted_long_term_reward = 0
            weighted_market_reward = 0
            weighted_risk_reward = 0
        else:
            weighted_financial_change = (financial_change / self.financial_resources) *
financial_change_weight
            weighted_long_term_reward = (long_term_reward / self.financial_resources) *
long_term_weight
            weighted_market_reward = (market_reward / self.financial_resources) * market_weight
            weighted_risk_reward = (risk_reward / self.financial_resources) * risk_weight

        # Calculate the total normalized weighted financial reward
```

```
        normalized_weighted_total_reward_financial = np.tanh(weighted_financial_change +
weighted_long_term_reward  + weighted_market_reward + weighted_risk_reward)


        # Calculate normalized human and social rewards

        if self.human_resources < 0:

            normalized_human_change = 0

        else:

            normalized_human_change = np.tanh(human_change / self.human_resources)


        if len(self.social_resources) == 0:

            normalized_social_reward = 0

        else:

            normalized_social_reward = np.tanh(social_reward / len(self.social_resources))


        # Calculate the new states after the action

        new_state_financial = self.financial_resources + financial_change

        new_state_human = self.human_resources + human_change

        new_state_social = len(self.social_resources) + social_reward


        # Add the new state and reward to the experience replay buffer for future learning

        self.add_to_buffer(old_state_financial, self.action,
normalized_weighted_total_reward_financial, new_state_financial, reward_type="financial")

        self.add_to_buffer(old_state_human, self.action, normalized_human_change, new_state_human,
reward_type="human")

        self.add_to_buffer(old_state_social, self.action, normalized_social_reward, new_state_social,
reward_type="social")


    def resolve_reward(self):

        """

        Resolve the reward for the agent based on the current actions and state.


        This method processes the agent's experience replay buffer to apply delayed rewards or
penalties

        based on previous actions. It updates the agent's financial, human, and social resources and

        also updates the Q-tables for learning. The method handles different types of rewards:
'financial',

        'human', and 'social', and applies them after a certain delay.


        Returns:

        --------

        None

        """
```

```
        # Loop through the experience replay buffer to process delayed rewards or penalties
        for entry in self.combined_buffer:
            # Unpack the buffer entry into its components
            old_state, action, reward, new_state, delay, reward_type = entry

            # Check if the delay has elapsed
            if delay < 1:
                # Apply the reward or penalty based on its type
                if reward_type == "financial":
                    # Update financial resources based on the new state and learn from the experience
                    self.financial_resources += new_state - old_state
                    self.learn(old_state, action, reward, new_state, self.q_table_financial,
"financial")

                elif reward_type == "human":
                    # Update human resources based on the new state and learn from the experience
                    self.human_resources += new_state - old_state
                    self.learn(old_state, action, reward, new_state, self.q_table_human, "human")

                elif reward_type == "social":
                    # Update social resources based on the new state and learn from the experience
                    if (new_state - old_state) > 0:
                        self.add_social_resource()
                    elif (new_state - old_state) < 0:
                        self.remove_social_resource()
                    self.learn(old_state, action, reward, new_state, self.q_table_social, "social")

                # Remove the processed entry from the buffer
                self.combined_buffer.remove(entry)
            else:
                # If the delay has not elapsed, decrement the delay counter
                updated_entry = (old_state, action, reward, new_state, delay - 1, reward_type)
                index = self.combined_buffer.index(entry)
                self.combined_buffer[index] = updated_entry # Update the entry in the buffer

    def get_dynamic_multipliers(self):
        """
        Retrieve dynamic multipliers that affect the agent's decision-making based on the current
market condition.
```

```
        Returns:
        --------
        tuple
            A tuple containing two float values: the first is the investment multiplier, and the second
is the saving multiplier.
        """
        # Initialize thresholds and multipliers from the agent's attributes
        booming_threshold = self.booming_threshold
        stagnating_threshold = self.stagnating_threshold
        investment_multiplier = self.investment_multiplier
        saving_multiplier = self.saving_multiplier

            # Determine the state of the market and adjust multipliers accordingly
        if self.model.market_condition > booming_threshold:
            # Market is booming, so return the original multipliers
            return investment_multiplier, saving_multiplier
        elif self.model.market_condition < stagnating_threshold:
            # Market is stagnating or shrinking, so swap the multipliers
            return saving_multiplier, investment_multiplier
        else:
            # Market is neutral, so average the multipliers
            return (investment_multiplier + saving_multiplier) / 2, (investment_multiplier +
saving_multiplier) / 2


    def compute_long_term_reward_with_compound_interest(self):
        """
        Compute the long-term reward with compound interest.

        Parameters:
        -----------
        reward : float
            The initial reward.
        interest_rate : float
            The interest rate for compounding.
        time_period : int
            The time period over which to compound the reward.

        Returns:
        --------
        float
```

```
            The compounded long-term reward.
        """
        # Retrieve the dynamic multipliers based on the current market condition
        investment_multiplier, saving_multiplier = self.get_dynamic_multipliers()

        # Calculate the compounded long-term reward based on the action taken
        if self.action == "invest":
            # If the action was to invest, use the investment multiplier
            return (1 + investment_multiplier)**self.consecutive_decisions_count - 1
        else:
            # If the action was to save, use the saving multiplier
            return (1 + saving_multiplier)**self.consecutive_decisions_count - 1

    def update_control_factor(self):
        """
        Update the control factor for the agent based on its exchange history.

        The control factor is calculated as the ratio of positive exchanges to the total number of
agents.
        A positive exchange is defined as one where the number of successful exchanges with a partner
        is greater than the number of refused exchanges.

        Returns:
        --------
        float
            The computed long-term reward based on compound interest and the number of consecutive
decisions.
        """
        # Get the total number of agents in the model
        total_agent_count = len(model.schedule.agents)

        # Initialize a counter for agents with whom this agent has had a net positive exchange history
        positive_agent_count = 0

        # Iterate through the exchange history
        for partner_id, history in self.exchange_history.items():
            # Calculate the net number of exchanges with each partner
            net_exchanges = history['successful'] - history['refused']

            # If the net number of exchanges is positive, increment the positive_agent_count
```

```
            if net_exchanges > 0:
                positive_agent_count += 1

        # Calculate the control factor as the ratio of positive_agent_count to total_agent_count
        # Handle the case where total_agent_count is 0 to avoid division by zero
        if total_agent_count == 0:
            self.control_factor = 0
        else:
            self.control_factor = positive_agent_count / total_agent_count  + 0.3


    def step(self):
        """
        Perform one step of the agent's set of actions.

        This method orchestrates the agent's actions within a single time step in the simulation.
        It includes choosing an action, resolving opportunities, updating history, interacting with
other agents,
        updating the control factor, resolving rewards, and observing the market.

        Returns:
        --------
        None
        """
        # Step 1: Normalize the agent's resources to a standard scale
        self.normalize_resources()

        # Step 2: Choose an action ("invest" or "save") based on Q-Learning and ε-greedy algorithms
        self.choose_action()

        # Step 3: Resolve a business opportunity based on the chosen action
        if self.resolve_a_business_opportunity:
            self.resolve_opportunity()

        # Step 4: Update the agent's history
        self.update_history()

        # Step 5: Re-normalize the agent's resources after resolving the opportunity
        self.normalize_resources()

        # Step 6: Interact and exchange resources with other agents
```

```
        # This represents the "crazy quilt principle" where resources are pieced together from various
sources
        if self.interact_and_exchange_resources_with_other_agents:
            self.interact_and_exchange()

        # Step 7: Update the control factor for the agent
        # This represents the "lemonade principle" where the agent adapts to unexpected events
        self.update_control_factor()

        # Step 8: Resolve rewards or penalties from earlier actions
        self.resolve_reward()

        # Step 9: Update the agent's history again after all actions have been resolved
        self.update_history()

        # Step 10: Observe the market to adjust loss tolerance
        self.observe_market()


class EntrepreneurModel(Model):
    """
    Represents the simulation model for a group of entrepreneurial agents.

    Attributes:
    -----------
    num_agents : int
        The number of agents in the model.
    schedule : RandomActivation
        The schedule for agent activation.
    market_condition : float
        The current market condition, represented as a float between 0.35 and 0.65.
    current_id : int
        The current ID to be assigned to a new agent.
    unexpected_events_log : list
        A list of tuples representing unexpected events. Each tuple is in the form (step, event_type).
    num_retired_agents : int
        The number of agents that have retired.
    datacollector : DataCollector
        A DataCollector object for collecting simulation data.
    unexpected_events : bool
        Whether unexpected events are activated or deactivated.
```

```
stable_market_condition : bool
    Whether the market condition is stable or not.
remove_agents_with_insufficient_resources : bool
    Whether to remove agents with insufficient resources.
agents_fixed_resources : bool
    Whether agents have fixed resources.
custom_resources_agent_100 : bool
    Whether agent 100 has custom resources.
market_change_rate : float
    The rate at which the market condition changes.
event_chance : float
    The chance of an unexpected event occurring.
market_crash_chance : float
    The chance of a market crash occurring.
market_crash_magnitude : float
    The magnitude of a market crash.
market_boom_magnitude : float
    The magnitude of a market boom.
breakthrough_chance : float
    The chance of a technological breakthrough.
human_resource_boost : float
    The boost in human resources due to a technological breakthrough.
social_event_chance : float
    The chance of a social event occurring.
agent_100_financial_resources : int
    The financial resources for agent 100.
agent_100_human_resources : int
    The human resources for agent 100.
agent_100_social_resources : int
    The social resources for agent 100.
agent_financial_resources : int
    The financial resources for all agents when fixed.
agent_human_resources : int
    The human resources for all agents when fixed.
agent_social_resources : int
    The social resources for all agents when fixed.
agent_random_financial_resources : float
    The financial resources for all agents when random.
agent_random_human_resources : float
```

```
    The human resources for all agents when random.
agent_random_social_resources : int
    The social resources for all agents when random.
"""

def __init__(self, num_agents):
    """
    Initialize a new EntrepreneurModel instance.

    Parameters:
    -----------
    num_agents : int
        The number of agents to include in the model.

    Returns:
    --------
    None
    """
    self._next_id = 0  # Initialize the _next_id attribute
    self.num_agents = num_agents
    self.schedule = RandomActivation(self)
    self.market_condition = random.uniform(0.35,0.65)
    self.current_id = 0
    self.unexpected_events_log = []  # Stores tuples in the form (step, event_type)
    self.num_retired_agents = 0
    self.datacollector = DataCollector( # Add DataCollector
        model_reporters={"MarketCondition": "market_condition"},
        agent_reporters={"FinancialResources": "financial_resources",
                         "HumanResources": "human_resources",
                         "SocialResources": lambda a: len(a.social_resources),
                         "LossTolerance": "loss_tolerance",
                         "ControlFactor": "control_factor"}  # Agent-level count of type}
    )

    # Activate or deactivate certain conditions for testing purposes
    self.unexpected_events = True
    self.stable_market_condition = False
    self.remove_agents_with_insufficient_resources = True
    self.agents_fixed_resources = True
```

```
self.custom_resources_agent_100 = False

# Parameters - update market condition
self.market_change_rate = 0.02

# Parameters - unexpected market event
self.event_chance = 0.015
self.market_crash_chance = 0.5
self.market_crash_magnitude = random.uniform(0.05,0.1)
self.market_boom_magnitude = random.uniform(0.05,0.1)

# Parameters - technological breakthrough
self.breakthrough_chance = 0.015
self.human_resource_boost = random.uniform(0.10, 0.30)  # increase human resources by 10 - 30%

# Parameters - technological breakthrough
self.social_event_chance = 0.04

# Agent 100 custom starting resources
self.agent_100_financial_resources =  150
self.agent_100_human_resources =  150
self.agent_100_social_resources = 2

# All agents starting resources when fixed
if self.agents_fixed_resources:
    self.agent_financial_resources = 100
    self.agent_human_resources = 100
    self.agent_social_resources = 1
# All agents starting resources when random
else:
    self.agent_random_financial_resources = self.random.uniform(50, 150)
    self.agent_random_human_resources = self.random.uniform(50, 150)
    self.agent_random_social_resources = random.randint(1, 3)



# Create agents
for i in range(self.num_agents):
    self.initialize_agent(self.next_id())
```

```
def step(self):
    """
    Advance the model by one step.

    This method performs a series of actions that update the state
    of the model and agents. It also collects data, removes agents
    with insufficient resources, triggers unexpected events and
    updates market conditions depending on various flags and settings.

    Returns:
    --------
    None
    """

    # Collect data before taking a step
    self.datacollector.collect(self)

    # Remove agents with insufficient financial or human resources
    if self.remove_agents_with_insufficient_resources:
        self.remove_agents()

    # Calling the unexpected events
    if self.unexpected_events:
        self.unexpected_market_event()
        self.technological_breakthrough()
        self.unexpected_social_event()

    # Update the market condition based on the actions of the agents
    if not self.stable_market_condition:
        self.update_market_condition()

    # Step through the schedule
    self.schedule.step()

def remove_agents_based_on_resource(self, resource_type):
    """
    Remove agents from the model based on a specific resource type.

    This method identifies agents with less than 1 unit of the specified resource type
```

```
        and removes them from the model. It also updates the social resources of other agents
        to reflect the removal.

        Parameters:
        -----------
        resource_type : str
            The type of resource to check for removal criteria (e.g., "financial_resources",
"human_resources").

        Returns:
        --------
        None
        """
        # Identify agents to remove based on the resource type
        agents_to_remove = [agent for agent in self.schedule.agents if getattr(agent, resource_type) <
1]

        if agents_to_remove:
            # Update the social resources of other agents
            for agent in self.schedule.agents:
                for agent_to_remove in agents_to_remove:
                    if agent_to_remove in agent.social_resources:
                        agent.remove_social_resource(agent_to_remove)

            # Remove the agents from the schedule and update counters
            for agent_to_remove in agents_to_remove:
                # print(f"Agent {agent_to_remove.unique_id} removed.")
                self.remove_agent_from_model(agent_to_remove.unique_id)
                self.schedule.remove(agent_to_remove)
                self.num_agents -= 1
                self.num_retired_agents += 1

    def remove_agents(self):
        """
        Remove agents from the model based on their financial and human resources.

        This method calls the `remove_agents_based_on_resource` method twice, once for
        financial resources and once for human resources, to remove agents who have less
        than 1 unit of either resource.
```

```
        Returns:
        --------
        None
        """
        # Remove agents based on financial resources
        self.remove_agents_based_on_resource('financial_resources')

        # Remove agents based on human resources
        self.remove_agents_based_on_resource('human_resources')

    def remove_agent_from_model(self, agent_to_remove_id):
        """
        Remove a specific agent from the model based on their unique ID.

        Parameters:
        -----------
        agent_id : int
            The unique ID of the agent to be removed.

        Returns:
        --------
        None
        """
        for agent in self.schedule.agents:
            if agent_to_remove_id in agent.exchange_history:
                del agent.exchange_history[agent_to_remove_id]

    def update_market_condition(self):
        """
        Update the market condition based on agent actions and market saturation.

        This method calculates the market condition by considering the proportion of agents
        who chose to invest and the market saturation, which is the ratio of successful exchanges
        to total exchanges. The market condition is then updated using these factors and a
        predefined market change rate.

        Returns:
        --------
        None
```

```
        """
        # Retrieve the market change rate
        market_change_rate = self.market_change_rate

        # Count the total number of agents and those who chose to invest
        total_agents = len(self.schedule.agents)
        investing_agents = len([agent for agent in self.schedule.agents if agent.action == 'invest'])

        # Calculate the number of successful and total exchanges for this step
        successful_exchanges_this_step = sum(agent.current_successful_exchanges for agent in
self.schedule.agents)
        total_exchanges_this_step = sum(agent.current_total_exchanges for agent in
self.schedule.agents)

        # Calculate market saturation
        if total_exchanges_this_step == 0:
            market_saturation = 0.0
        else:
            market_saturation = successful_exchanges_this_step / total_exchanges_this_step

        # Update the market condition
        self.market_condition += ((investing_agents / total_agents - 0.5) * 0.5  + (market_saturation -
0.5) * 0.5) * market_change_rate

        # Ensure the market condition stays within [0, 1]
        self.market_condition = max(0, min(1, self.market_condition))

    def initialize_agent(self, agent_id):
        """
        Initialize an agent with either custom or default resources.

        This method initializes an agent with either custom resources (if the agent_id is 100 and
        custom_resources_agent_100 is True) or default resources. The agent is then added to the
        schedule, and the agents dictionary is updated.

        Parameters:
        -----------
        agent_id : int
            The unique identifier for the agent to be initialized.

        Returns:
```

```
        --------

        None

        """

        # Initialize an agent with custom resources if agent_id is 100 and custom_resources_agent_100
is True

        if agent_id == 100 and self.custom_resources_agent_100:

            agent = EntrepreneurAgent(agent_id, self)

            agent.financial_resources = self.agent_100_financial_resources

            agent.human_resources = self.agent_100_human_resources

            agent.social_resources = []

            for i in range(self.agent_100_social_resources):

                agent.add_social_resource()

            self.schedule.add(agent)

        # Initialize all other agents with default resources

        else:

            agent = EntrepreneurAgent(agent_id, self)

            agent.financial_resources = self.agent_financial_resources if self.agents_fixed_resources
else self.agent_random_financial_resources

            agent.human_resources = self.agent_human_resources if self.agents_fixed_resources else
self.agent_random_human_resources

            agent.social_resources = []

            for i in range(self.agent_social_resources if self.agents_fixed_resources else
self.agent_random_social_resources):

                agent.add_social_resource()

            self.schedule.add(agent)


        # Update the agents dictionary

        self.agents = {agent.unique_id: agent for agent in self.schedule.agents}


    def unexpected_market_event(self):

        """

        Simulate an unexpected market event, either a market crash or a market boom.


        This method simulates an unexpected market event based on the given probabilities.

        It either triggers a market crash or a market boom, affecting the market condition

        and causing agents to react to the surprise. The event is then logged.


        Returns:

        --------

        None

        """
```

```
        event_chance = self.event_chance
        market_crash_chance = self.market_crash_chance
        market_crash_magnitude = self.market_crash_magnitude
        market_boom_magnitude = self.market_boom_magnitude

        # Chance of an unexpected market event occurring
        if random.random() < event_chance:
            if random.random() < market_crash_chance:
                self.market_condition -= market_crash_magnitude  # Trigger market crash
                for agent in self.schedule.agents:  # Agents react to the market crash
                    agent.react_to_surprise("market_crash")
                self.unexpected_events_log.append((self.schedule.steps, "market crash")) # Log the
event
            else:
                self.market_condition += market_boom_magnitude  # Trigger market boom
                for agent in self.schedule.agents:  # Agents react to the market boom
                    agent.react_to_surprise("market_boom")
                self.unexpected_events_log.append((self.schedule.steps, "market boom")) # Log the event

            # Ensure the market condition stays within the range [0, 1]
            self.market_condition = max(0, min(1, self.market_condition))  # Ensure that the value
remains between 0 and 1

    def technological_breakthrough(self):
        """
        Simulate a technological breakthrough that boosts the human resources of a randomly chosen
agent.

        This method simulates a technological breakthrough based on the given probability.
        It selects a random agent and increases their human resources. Other agents in the
        model react to this unexpected event. The event is then logged.

        Returns:
        --------
        None
        """
        breakthrough_chance = self.breakthrough_chance
        human_resource_boost = self.human_resource_boost

        # Chance of a technological breakthrough occurring
        if random.random() < breakthrough_chance and self.schedule.agents:
```

```
            # Select a random agent to receive the boost in human resources
            agent = random.choice(self.schedule.agents)

            # Increase the human resources of the selected agent
            agent.human_resources = agent.human_resources * (1 + human_resource_boost)

            # Other agents react to the technological breakthrough
            for other_agent in self.schedule.agents:  # agents react to surprise
                if other_agent != agent:
                    other_agent.react_to_surprise("technological_breakthrough")

            # Log the technological breakthrough event
            self.unexpected_events_log.append((self.schedule.steps, "technological breakthrough"))

    def unexpected_social_event(self):
        """
        Simulate an unexpected social event that boosts the social resources of a randomly chosen
agent.

        This method simulates an unexpected social event based on the given probability.
        It selects a random agent and increases their social resources by adding a random
        number of new social resources. Other agents in the model react to this unexpected event.
        The event is then logged.

        Returns:
        --------
        None
        """
        social_event_chance = self.social_event_chance

        # Chance of an unexpected social event occurring
        if random.random() < social_event_chance:
            # Select a random agent to receive the boost in social resources
            agent = random.choice(self.schedule.agents)

            # Determine the number of new social resources to add
            agents_to_add =  random.randint(1, 5)

            # Add the new social resources to the selected agent
            for i in range(agents_to_add):
```

```
                agent.add_social_resource()

            # Other agents react to the unexpected social event
            for other_agent in self.schedule.agents:
                other_agent.react_to_surprise("social_event")

            # Log the unexpected social event
            self.unexpected_events_log.append((self.schedule.steps, "social event"))

    def get_unexpected_events_info(self):
        """
        Retrieve information about unexpected events that have occurred in the model.

        This method returns the number of unexpected events that have occurred so far
        and a log of these events.

        Returns:
        --------
        tuple
            A tuple containing the number of unexpected events and the log of these events.
        """
        # Count the number of unexpected events
        num_events = len(self.unexpected_events_log)

        # Return the number of events and the log
        return num_events, self.unexpected_events_log

    def reset_datacollector(self):
        """
        Reset the DataCollector object to its initial state.

        This method clears all the collected data from the DataCollector object,
        effectively resetting it to its initial state.

        Returns:
        --------
        None
        """
        # Reset model-level variables
```

```
        self.datacollector.model_vars = defaultdict(list)

        # Reset agent-level variables
        self.datacollector.agent_vars = defaultdict(lambda: defaultdict(list))

        # Reset tables
        self.datacollector.tables = defaultdict(lambda: pd.DataFrame())

    def replicate(self, num_replications, num_steps):
        """
        Replicate the model by running it multiple times with different initial conditions or
parameters.

        Parameters:
        -----------
        num_replications : int
            The number of times the model should be replicated.

        num_steps : int
            The number of steps each replication should run for.

        Returns:
        --------
        list
            A list containing the results of each replication. Each entry in the list is a tuple
containing
            lists of financial resources, human resources, and social resources for all agents at the
end of
            each replication.
        """
        results = [] # List to store the results of each replication

        # Loop through the number of replications
        for x in range(num_replications):
            self.schedule = RandomActivation(self)  # Reset the schedule
            self.market_condition = random.uniform(0.35,0.65) # Reset the market condition
            self.next_id(reset=True)  # Reset the agent ID counter

            # Create agents
            for i in range(self.num_agents):
                self.initialize_agent(self.next_id())
```

```
            # Run the model for a given number of steps
            for i in range(num_steps):
                self.step()

            # Get Agent 100 at the end of the run
            agent_100 = next((agent for agent in self.schedule.agents if agent.unique_id == 100), None)
            if agent_100 == None:
                print(f"Agent 100 has been removed due to insufficient resources in replication {x}.
Run does not count.")
                continue

            # Collect the resources of all agents at the end of the run
            financial_resources = [agent.financial_resources for agent in self.schedule.agents]
            human_resources = [agent.human_resources for agent in self.schedule.agents]
            social_resources = [len(agent.social_resources) for agent in self.schedule.agents]
            loss_tolerance = [agent.loss_tolerance for agent in self.schedule.agents]
            control_factor = [agent.control_factor for agent in self.schedule.agents]
            agent_100_financial_resources = [agent_100.financial_resources]
            agent_100_human_resources = [agent_100.human_resources]
            agent_100_social_resources = [len(agent_100.social_resources)]
            agent_100_loss_tolerance = [agent_100.loss_tolerance]
            agent_100_control_factor = [agent_100.control_factor]

            # Append the results to the results list
            results.append((financial_resources, human_resources, social_resources, loss_tolerance,
control_factor, agent_100_financial_resources, agent_100_human_resources, agent_100_social_resources,
agent_100_loss_tolerance, agent_100_control_factor))
        return results


    def next_id(self, reset=False):
        """
        Generate the next unique ID for an agent. Optionally reset the ID counter.

        Parameters:
        -----------
        reset : bool, optional
            Whether to reset the ID counter. Default is False.

        Returns:
        --------
```

```
        int
            The next unique ID.
        """

        # Reset the ID counter if specified
        if reset:
            self._next_id = -1

        # Increment the ID counter
        self._next_id += 1

        return self._next_id

    def print_replication(self, num_replications, num_steps):
        """
        Run the model for a given number of replications and steps, then print the results.

        Parameters:
        -----------
        num_replications : int
            The number of times the model should be replicated.
        num_steps : int
            The number of steps each replication should run for.

        Returns:
        --------
        None
        """

        # Reset the counter for retired agents and the data collector
        self.reset_datacollector()

        # Run the model for the specified number of replications and steps
        results = self.replicate(num_replications, num_steps)

        # Process the results to separate them by resource type
        financial_resources_results = [res[0] for res in results]
        human_resources_results = [res[1] for res in results]
        social_resources_results = [res[2] for res in results]
```

```
        loss_tolerance_results = [res[3] for res in results]
        control_factor_results = [res[4] for res in results]
        agent_100_financial_resources_results = [res[5] for res in results]
        agent_100_human_resources_results = [res[6] for res in results]
        agent_100_social_resources_results = [res[7] for res in results]
        agent_100_loss_tolerance_results = [res[8] for res in results]
        agent_100_control_factor_results = [res[9] for res in results]

        # Calculate mean and standard deviation for each resource type
        financial_resources_mean = np.mean([np.mean(res) for res in financial_resources_results])
        financial_resources_std = np.mean([np.std(res) for res in financial_resources_results])
        human_resources_mean = np.mean([np.mean(res) for res in human_resources_results])
        human_resources_std = np.mean([np.std(res) for res in human_resources_results])
        social_resources_mean = np.mean([np.mean(res) for res in social_resources_results])
        social_resources_std = np.mean([np.std(res) for res in social_resources_results])
        loss_tolerance_mean =np.mean([np.mean(res) for res in loss_tolerance_results])
        loss_tolerance_std = np.mean([np.std(res) for res in loss_tolerance_results])
        control_factor_mean = np.mean([np.mean(res) for res in control_factor_results])
        control_factor_std = np.mean([np.std(res) for res in control_factor_results])
        agent_100_financial_resources_mean = np.mean([np.mean(res) for res in
agent_100_financial_resources_results])
        agent_100_financial_resources_std = np.std(agent_100_financial_resources_results)
        agent_100_human_resources_mean = np.mean([np.mean(res) for res in
agent_100_human_resources_results])
        agent_100_human_resources_std = np.std(agent_100_human_resources_results)
        agent_100_social_resources_mean = np.mean([np.mean(res) for res in
agent_100_social_resources_results])
        agent_100_social_resources_std = np.std(agent_100_social_resources_results)
        agent_100_loss_tolerance_mean =np.mean([np.mean(res) for res in
agent_100_loss_tolerance_results])
        agent_100_loss_tolerance_std = np.std(agent_100_loss_tolerance_results)
        agent_100_control_factor_mean = np.mean([np.mean(res) for res in
agent_100_control_factor_results])
        agent_100_control_factor_std = np.std(agent_100_control_factor_results)

        # Print the results
        print(f'---------- Results of replicated simulation ----------')
        print(f"Replications of model: {num_replications}")

        # Print unexpected events
        num_events, event_log = self.get_unexpected_events_info()
        print(f"Number of unexpected events: {num_events}")
```

```
        for step, event_type in event_log:
            print(f"At step {step}, unexpected event of type {event_type} occurred.")



        # Create a DataFrame to display the results
        results = pd.DataFrame({
            'Parameter Type': ['Financial resources', 'Human resources', 'Social resources', 'Loss tolerance', 'Control factor'],
            'Agent 100 Mean': [agent_100_financial_resources_mean, agent_100_human_resources_mean, agent_100_social_resources_mean, agent_100_loss_tolerance_mean, agent_100_control_factor_mean],
            'Agent 100 Standard Deviation': [agent_100_financial_resources_std, agent_100_human_resources_std, agent_100_social_resources_std, agent_100_loss_tolerance_std, agent_100_control_factor_std],
            'Mean': [financial_resources_mean, human_resources_mean, social_resources_mean, loss_tolerance_mean, control_factor_mean],
            'Standard Deviation': [financial_resources_std, human_resources_std, social_resources_std, loss_tolerance_std, control_factor_std]
        })

        # Print the DataFrame
        print(results)
        # self.t_test("financial resources", [np.mean(res) for res in agent_100_financial_resources_results], [np.mean(res) for res in financial_resources_results])
        # self.t_test("human resources", [np.mean(res) for res in agent_100_human_resources_results], [np.mean(res) for res in human_resources_results])


    def t_test(self, resource_type, agent_100_results, all_agents_results):
        # Print statistical tests
        print(f"Statistical t-tests for {resource_type} means")


        # t-Test durchführen
        t_stat, p_value = stats.ttest_ind(agent_100_results, all_agents_results)

        print(f"t-statistics: {t_stat}, p-value: {p_value}")


        # p-Wert überprüfen
        alpha = 0.05  # Significance level
        if p_value < alpha:
            print(f"There is a statistically significant difference between agent 100 and the average of all agents {resource_type}.")
        else:
            print(f"There is no statistically significant difference between agent 100 and the average of all agents {resource_type}.")
```

```
def interpret(self):
    """
    Interpret the results of the simulation by analyzing various metrics and statistics.

    Returns:
    --------
    None
    """
    # Collect various metrics from the agents
    counter_invest_list = [agent.counter_invest for agent in self.schedule.agents]
    counter_save_list = [agent.counter_save for agent in self.schedule.agents]
    financial_resources = [agent.financial_resources for agent in self.schedule.agents]
    human_resources = [agent.human_resources for agent in self.schedule.agents]
    social_resources = [len(agent.social_resources) for agent in self.schedule.agents]
    loss_tolerance_list = [agent.loss_tolerance for agent in self.schedule.agents]
    control_factor_list = [agent.control_factor for agent in self.schedule.agents]

    # Calculate aggregate and statistical measures
    counter_invest_all = sum(counter_invest_list)
    counter_save_all = sum(counter_save_list)
    mean_financial_resources = np.mean(financial_resources)
    std_financial_resources = np.std(financial_resources)
    mean_human_resources = np.mean(human_resources)
    std_human_resources = np.std(human_resources)
    mean_social_resources = np.mean(social_resources)
    std_social_resources = np.std(social_resources)
    mean_loss_tolerance = np.mean(loss_tolerance_list)
    std_loss_tolerance = np.std(loss_tolerance_list)
    mean_control_factor = np.mean(control_factor_list)
    std_control_factor = np.std(control_factor_list)

    # Print the results
    print(f'---------- Results of simulation ----------')
    print(f"# Investments: {counter_invest_all} & # Saves: {counter_save_all}")
    print(f"# Retired agents due to insufficient financial resources: {self.num_retired_agents}")

    # Print unexpected events
    num_events, event_log = self.get_unexpected_events_info()
```

```
        print(f"Number of unexpected events: {num_events}")

        for step, event_type in event_log:

            print(f"At step {step}, unexpected event of type {event_type} occurred.")


        # Special case for Agent 100

        agent_100 = next((agent for agent in self.schedule.agents if agent.unique_id == 100), None)

        if agent_100 == None:

            print("Agent 100 has been removed due to insufficient financial resources.")

            # Create a DataFrame to display the results

            results = pd.DataFrame({

                'Parameter Type': ['Financial resources', 'Human resources', 'Social resources', 'Loss
Tolerance', 'Control Factor'],

                'Mean': [mean_financial_resources, mean_human_resources, mean_social_resources,
mean_loss_tolerance, mean_control_factor],

                'Standard Deviation': [std_financial_resources, std_human_resources,
std_social_resources, std_loss_tolerance, std_control_factor]

            })

        else:

            print(f"Exchange history of agent {agent_100.unique_id}:")

            for partner_id, history in agent_100.exchange_history.items():

                print(f"  With agent {partner_id}: {history['successful']} successful exchanges,
{history['refused']} refused exchanges")

            # Create a DataFrame to display the results

            results = pd.DataFrame({

                'Parameter Type': ['Financial resources', 'Human resources', 'Social resources', 'Loss
Tolerance', 'Control Factor'],

                'Agent 100': [agent_100.financial_resources, agent_100.human_resources,
len(agent_100.social_resources), agent_100.loss_tolerance, agent_100.control_factor],

                'Mean': [mean_financial_resources, mean_human_resources, mean_social_resources,
mean_loss_tolerance, mean_control_factor],

                'Standard Deviation': [std_financial_resources, std_human_resources,
std_social_resources, std_loss_tolerance, std_control_factor]

            })


        # Print the DataFrame

        print(results)


    def visualize(self, agent_id):

        """

        Visualize the distribution of resources among the agents, highlighting the resources of a
specific agent.


        Parameters:
```

```
        -----------
        agent_id : int
            The unique ID of the agent to be highlighted in the visualization.

        Returns:
        --------
        None
        """

        # Collect resource data from all agents
        financial_resources = [agent.financial_resources for agent in self.schedule.agents]
        human_resources = [agent.human_resources for agent in self.schedule.agents]
        social_resources = [len(agent.social_resources) for agent in self.schedule.agents]

        # Find the specific agent by ID
        agent_100 = next((agent for agent in self.schedule.agents if agent.unique_id == agent_id),
None)
        if agent_100 is None:
            print(f"Agent {agent_id} has been removed due to insufficient financial resources.")

        # Create a figure with three subplots
        fig, axs = plt.subplots(1, 3, figsize=(15, 5))

        # Plot financial resources
        n, bins, patches = axs[0].hist(financial_resources, alpha=0.8, color='#00B2EE', label='All
Agents')
        if agent_100:
            bin_index = np.digitize(agent_100.financial_resources, bins) - 1
            if bin_index < len(patches):
                patches[bin_index].set_facecolor('#EE4000')
        axs[0].set_title('Distribution of financial resources')
        axs[0].set_xlabel('Financial resources')
        axs[0].set_ylabel('Number of agents')

        # Plot human resources
        n, bins, patches = axs[1].hist(human_resources, alpha=0.8, color='#00B2EE', label='All Agents')
        if agent_100:
            bin_index = np.digitize(agent_100.human_resources, bins) - 1
            if bin_index < len(patches):
                patches[bin_index].set_facecolor('#EE4000')
```

```
        axs[1].set_title('Distribution of human resources')
        axs[1].set_xlabel('Human resources')
        axs[1].set_ylabel('Number of agents')

        # Plot social resources
        n, bins, patches = axs[2].hist(social_resources, alpha=0.8, color='#00B2EE', label='All
Agents')
        if agent_100:
            bin_index = np.digitize(len(agent_100.social_resources), bins) - 1
            if bin_index < len(patches):
                patches[bin_index].set_facecolor('#EE4000')
        axs[2].set_title('Distribution of social resources')
        axs[2].set_xlabel('Number of contacts')
        axs[2].set_ylabel('Number of agents')

        # Add legends and titles
        legend_elements = [Patch(facecolor='#00B2EE', edgecolor='#00B2EE', label='All Agents'),
                           Patch(facecolor='#EE4000', edgecolor='#EE4000', label=f'Agent {agent_id}')]
        for ax in axs:
            ax.legend(handles=legend_elements)

        # Add a super title for the entire figure
        if agent_100:
            fig.suptitle(f'Distribution of resources for all agents compared to Agent {agent_id}')
        else:
            fig.suptitle(f'Distribution of resources for all agents, Agent {agent_id} has retired due
to insufficient financial resources')

        # Adjust layout and show the plot
        plt.tight_layout()
        plt.show()

    def print_resources_for_agent(self, agent_id):
        """
        Visualize and print the resources for a specific agent over time.

        Parameters:
        -----------
        agent_id : int
            The unique identifier of the agent whose resources are to be visualized.
```

```
        Returns:
        --------
        None
        """
        agent_exists = False

        # Retrieve data for all agents from the data collector
        agent_data = self.datacollector.get_agent_vars_dataframe()
        model_data = self.datacollector.get_model_vars_dataframe()

        # Check if the specified agent still exists in the model
        agent_ids = [agent.unique_id for agent in self.schedule.agents]
        if 100 not in agent_ids:
            pass # The agent has been removed, typically due to insufficient resources
        else:
            # Filter the data to only include information for the specified agent
            agent_100_data = agent_data.xs(agent_id, level="AgentID")
            agent_exists = True

        # Initialize the plot with 3 rows and 2 columns
        fig, axs = plt.subplots(3, 2, figsize=(15, 15))

        # Plotting Financial Resources Over Time
        # Red line for the specified agent, dashed blue line for the average across all agents
        if agent_exists:
            agent_100_data['FinancialResources'].plot(ax=axs[0, 0], color='red', label=f'Agent
{agent_id}')
        agent_data.groupby('Step')['FinancialResources'].mean().plot(ax=axs[0, 0], linestyle='--',
color='blue', label='Average')
        axs[0, 0].set_title('Financial Resources')
        axs[0, 0].set_ylabel('Financial Resources')
        axs[0, 0].set_xlabel('Step')
        axs[0, 0].legend()

        # Plotting Human Resources Over Time
        # Red line for the specified agent, dashed blue line for the average across all agents
        if agent_exists:
            agent_100_data['HumanResources'].plot(ax=axs[0, 1], color='red', label=f'Agent {agent_id}')
        agent_data.groupby('Step')['HumanResources'].mean().plot(ax=axs[0, 1], linestyle='--',
color='blue', label='Average')
```

```
        axs[0, 1].set_title('Human Resources')

        axs[0, 1].set_ylabel('Human Resources')

        axs[0, 1].set_xlabel('Step')

        axs[0, 1].legend()


        # Plotting Social Resources Over Time

        # Red line for the specified agent, dashed blue line for the average across all agents

        if agent_exists:

            agent_100_data['SocialResources'].plot(ax=axs[1, 0], color='red', label=f'Agent
{agent_id}')

        agent_data.groupby('Step')['SocialResources'].mean().plot(ax=axs[1, 0], linestyle='--',
color='blue', label='Average')

        axs[1, 0].set_title('Social Resources')

        axs[1, 0].set_ylabel('Social Resources')

        axs[1, 0].set_xlabel('Step')

        axs[1, 0].legend()


        # Plotting Loss Tolerance Over Time

        # Red line for the specified agent, dashed blue line for the average across all agents

        if agent_exists:

            agent_100_data['LossTolerance'].plot(ax=axs[1, 1], color='red', label=f'Agent {agent_id}')

        agent_data.groupby('Step')['LossTolerance'].mean().plot(ax=axs[1, 1], linestyle='--',
color='blue', label='Average')

        axs[1, 1].set_title('Loss Tolerance')

        axs[1, 1].set_ylabel('Loss Tolerance')

        axs[1, 1].set_xlabel('Step')

        axs[1, 1].legend()


        # Plotting Control Factor Over Time

        # Red line for the specified agent, dashed blue line for the average across all agents

        if agent_exists:

            agent_100_data['ControlFactor'].plot(ax=axs[2, 0], color='red', label=f'Agent {agent_id}')

        agent_data.groupby('Step')['ControlFactor'].mean().plot(ax=axs[2, 0], linestyle='--',
color='blue', label='Average')

        axs[2, 0].set_title('Control Factor')

        axs[2, 0].set_ylabel('Control Factor')

        axs[2, 0].set_xlabel('Step')

        axs[2, 0].legend()


        # Plot Market Condition Over Time

        # This assumes that the market condition is relevant for the agent
```

```
        last_x = model_steps
        model_data_tail = model_data.tail(last_x)
        model_data_tail['MarketCondition'].plot(ax=axs[2, 1], color='blue')
        axs[2, 1].set_title('Market Condition')
        axs[2, 1].set_ylabel('Market Condition')
        axs[2, 1].set_xlabel('Step')

        # Set the overall title for the plots
        if agent_exists:
            fig.suptitle(f'Resources for agent {agent_id} in comparison to average')
        else:
            fig.suptitle(f'Average resources for other agents, Agent {agent_id} has retired due to
insufficient financial resources')

        # Adjust layout to prevent overlap
        plt.tight_layout()
        plt.show()

################################ Start the model ################################################

start_model = True
start_replication = False

model_agents = 100
model_steps = 100
num_replication = 10

if start_model:
    # Initialize the model
    model = EntrepreneurModel(model_agents)

    # Run the model for 100 steps

    for i in range(model_steps):
        model.step()

    # Specify that we are interested in the agent with ID 100
    agent_id = 100

    # Analyze and print the results of the simulation
```

```
    model.interpret()

    # Generate visualizations for the simulation, focusing on agent with ID 100
    model.visualize(agent_id)

    # Print and plot the resources for the agent with ID 100 over time
    model.print_resources_for_agent(agent_id)

################################ Replicate the model ##########################################

if start_replication:

    # Initialize a second model for replication
    model = EntrepreneurModel(model_agents)

    # Run the replications and print the aggregated results
    model.print_replication(num_replication, model_steps)

    # Specify that we are interested in the agent with ID 100
    # agent_id = 100

    # Generate visualizations for the replicated model, focusing on agent with ID 100
    #model.visualize(agent_id)

    # Print and plot the resources for the agent with ID 100 over time in the replicated model
    #model.print_resources_for_agent(agent_id)
```