Universität des Saarlandes
Naturwissenschaftlich-Technische Fakultät I
Fachrichtung Informatik

Echtzeitentscheidungen durch Trajektoriensimulation in Dual Reality

Masterthesis

vorgelegt von

Christian Felix Bürckert

am 18.06.2014

Angefertigt und betreut unter der Leitung von
Gerrit Kahl

Begutachtet von
Prof. Dr. Antonio Krüger
Prof. Dr. Jörg Hoffmann

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, ...

(Datum / Date) (Unterschrift/ Signature)

Zusammenfassung

Bei der Monte-Carlo-Baumsuche werden, anstatt einer Heuristik, Monte-Carlo Simulationen verwendet, um die Suche zu leiten. Die Monte-Carlo-Baumsuche wird zur Entscheidungsfindung in diskreten Zustandsräumen wie zum Beispiel beim Schach oder der Wegsuche verwendet.

Eine Dual Reality ist eine beidseitige Verknüpfung der Realität mit einem virtuellen Modell dieser Realität. Dabei nehmen Sensoren die Realität wahr und das virtuelle Modell wird entsprechend dieser wahrgenommenen Veränderungen angepasst. Änderungen am virtuellen Modell werden, sofern möglich, durch Aktuatoren in die Realität übertragen. Die Zukunft des Modells einer Dual Reality kann mit Suchbäumen jedoch nicht ausgedrückt werden, da die zeitliche Entwicklung eine unendliche Verzweigung eines kontinuierlichen Modells darstellt.

Die in dieser Arbeit vorgestellte Trajektoriensimulation kann verwendet werden, um die Zukunft eines virtuellen Modells einer Realität einzuschätzen und ermöglicht dadurch das Fällen von Entscheidungen, die in Echtzeit durch Aktuatoren in die Wirklichkeit umgesetzt werden können. Trajektoriensimulationen sind diskrete Simulationen kontinuierlicher Trajektorien, die dazu verwendet werden, den Erwartungswert einer Bewertungsfunktion in der Zukunft zu approximieren. Basierend auf der Einschätzung des Erwartungswerts wird eine mathematische Relation eingeführt, die es ermöglicht, die Alternativen einer Entscheidung zu bewerten. Eine so gewonnene Entscheidung kann als rational bezeichnet werden, da sie versucht den größten Erwartungswert zu erzielen. Dabei sind Fehleinschätzungen, die durch Schwankungen bei der Erwartungswertapproximation entstehen, möglich. Die Trajektoriensimulation kann als kontinuierliche Erweiterung der Monte-Carlo-Baumsuche gesehen werden.

Zur Umsetzung wird in dieser Arbeit die Trajektoriensimulation zuerst mathematisch definiert und dann ein Rahmenwerk eingeführt, in dem Trajektoriensimulationen genutzt werden, um Entscheidungen in Dual Reality zu fällen. Dieses Rahmenwerk wird zusätzlich in drei Szenarien umgesetzt: Das erste Szenario ist eine intelligente Ampelsteuerung, die den Verkehrsfluss optimieren und die Wartezeiten der Autos, verglichen mit einer Zeitschaltung, drastisch verringert. Das zweite Szenario ist ein Kundenleitsystem für große Supermärkte, dass durch Kassenempfehlungen auf Monitoren oder Smartphones die Wartezeit an den Kassen verringern und die Auslastung der Kassen besser verteilen kann. Das dritte Szenario unterstützt Filialleiter bei der Entscheidung, wann welche Kasse geöffnet oder geschlossen werden soll, um eine gewünschte Wartezeitauslastung zu erzielen und zeigt, wie schnell neue Entscheidungen auf ein vorhandenes Modell umgesetzt werden können.

Der Vorteil, dass ein Computer dazu in der Lage ist viele Simulationen parallel durchzuführen, kann dazu genutzt werden, die Qualität der Entscheidungen zu verbessern und macht Trajektoriensimulation zu einer zukunftssicheren KI-Methode, deren Entscheidungsqualität mit zunehmender Anzahl der CPU-Kerne automatisch wächst.

Inhaltsverzeichnis

1 Einführung

Eine Entscheidung ist eine Wahl zwischen zwei oder mehreren Alternativen. Echtzeitentscheidungen sind Entscheidungen, die schnell gefällt werden müssen. Sie können zum Beispiel dadurch entschieden werden, dass man die verschiedenen Alternativen durchdenkt. Durchdenken bedeutet, man bewertet die Auswirkungen der Alternativen in der Zukunft. Dazu braucht man eine konkrete Vorstellung dieser Zukunft, die aus der Gegenwart gewonnen werden kann, sowie die konkrete Auswirkung der Alternative in dieser Zukunft. Diese Auswirkungen entwickeln sich durch Veränderungen, die in dieser Zukunft durchgeführt werden. Solche durchdachten Entscheidungen nennen wir rationale Entscheidungen. Werden die Entscheidungen nicht durchdacht, sondern zum Beispiel aus einem Bauchgefühl heraus gefällt, so spricht man von emotionalen oder spontanen Entscheidungen. Ferner gibt es noch die zufälligen Entscheidungen, die mit der Hilfe eines Zufallsprozesses gefällt werden.

Diese Arbeit widmet sich den rationalen Entscheidungen und setzt das Konzept, welches wir als Durchdenken bezeichnen, für einen Computer um. Wie angesprochen, braucht der Computer dafür ein konkretes Verständnis der Gegenwart und muss in der Lage sein, die Zukunft daraus abzuleiten. Eine Alternative spiegelt sich dann in der Veränderung dieser abgeleiteten Zukunft wieder. Um die Auswirkung der Veränderung bewerten zu können, muss diese Zukunft meist noch ein wenig weiter gedacht werden. Natürlich braucht man auch ein geeignetes Bewertungskriterium, da sonst der Nutzen aus dem Durchdenken nicht extrahiert werden kann.

Dual Reality beschäftigt sich mit der Synchronisation zwischen einer virtuellen Realität und der Realität [29, 30, 21]. Dazu werden Veränderungen in der Realität über Sensoren wahrgenommen und in der virtuellen Realität nachgebildet [60]. Änderungen der virtuellen Realität dagegen werden sofern möglich über Aktuatoren in der Realität umgesetzt oder Menschen führen diese Änderungen nachträglich durch. Dual Reality ermöglicht es also Entscheidungen umzusetzen und zusätzlich bildet es eine geeignete Methode um die Gegenwart in einem Computer abzubilden.

Entscheidungsunterstützungssysteme sind Computerprogramme, die Menschen bei der Findung von rationalen Entscheidungen durch die Bereitstellung von Informationen über die Zukunft unterstützen. Diese Systeme finden sich zum Beispiel im Investmentbanking [53], aber unter anderem auch in der Landwirtschaft [1]. Diese Informationen über die Zukunft können durch Simulationen gewonnen werden. Simulationen sind also eine geeignete Methode, um aus der Gegenwart die Zukunft abzuleiten. Die zeitliche Entwicklungslinie eines solchen dynamischen Systems bezeichnet man als Trajektorie. Diese können daher auch durch Simulationen gewonnen werden. Sie eignen sich zur mathematischen Bewertung einer Zukunft und sind somit das Kernkonstrukt dieser Arbeit.

Damit sind alle Komponenten spezifiziert, die für Echtzeitentscheidungen benötigt werden: Dual Reality wird verwendet, um die Gegenwart abzubilden und die Änderungen, die durch die Entscheidung entstehen sollen, in der Wirklichkeit zu realisieren.

Simulationen werden verwendet um die Zukunft der Gegenwart über Trajektorien für jede mögliche Alternative zu bestimmen. Sie beinhalten also die durch die Alternative spezifizierte Veränderung als eine Art Richtungsänderung. Bewertet man die so gewonnenen Trajektorien, so kann eine Entscheidung rational begründet werden.

1.1 Motivation

Computerprogramme fällen viele Entscheidungen und beeinflussen so das menschliche Verhalten: Sie steuern unsere Heizungen, unterstützen uns beim Autofahren (ABS, Navigationssysteme) und regeln unseren Verkehr. Den Entscheidungen liegen meist einfache Algorithmen zu Grunde. Eine Heizungssteuerung schaltet die Heizung ein, sobald ein Schwellenwert unterschritten und schaltet sie wieder aus, sobald ein Schwellenwert überschritten wurde. Ampeln folgen häufig simplen Zeitschaltungen, können aber auch an komplexe Verkehrscomputer mit vielen Sensoren angeschlossen sein. Diese versuchen sich dann intelligent dem Verkehr anzupassen [46]. Für solche komplexen Systeme werden KI-Systeme wie künstliche Neuronale Netze, Bayessche Netze, Planungssysteme bis hin zu Multiagentensystemen verwendet. Die meisten dieser Systeme basieren auf natürlichen Prozessen oder orientieren sich am Vorbild der menschlichen Entscheidungsfindung. Dabei könnte man die Entscheidungen, die durch Neuronale Netze gewonnen werden, als emotionale oder spontane Entscheidungen bezeichnen, da sie aufgrund von gelernten Erfahrungen die Auswirkung auf die Zukunft intuitiv bewerten. Bayessche Netze dagegen bewerten die Zukunft anhand von statistischen Modellen, sie berücksichtigen dabei die Auswirkungen auf die Zukunft nur indirekt. Der oben beschriebene Prozess der rationalen Entscheidungsfindung wird am ehesten noch von den Monte-Carlo-Simulationsbäumen durchlaufen [6]. Diese führen eine Bestensuche durch, die anstatt von einer Heuristik durch Simulationen geleitet wird. Diese Suchbäume funktionieren nur auf diskreten Modellen mit endlichen Zustandsräumen, die in der Realität jedoch selten auftreten. Daher werden die Monte-Carlo-Simulationsbäume auch hauptsächlich in rundenbasierten Computerspielen verwendet [4, 6, 58].

In dieser Arbeit soll erklärt werden, wie Echtzeitentscheidungen getroffen werden, die sich auf Aktuatoren in der Wirklichkeit beziehen. Als Grundlage dienen durch Dual Reality-Methoden gewonnene kontinuierliche Modelle. Diese kontinuierlichen Modelle werden mit Hilfe von Simulatoren in Trajektorien verschiedener Zukünfte des kontinuierlichen Raums überführt und spiegeln dort die Auswirkungen der verschiedenen Alternativen einer Entscheidung wieder. Durch die Bewertung dieser Trajektorien wird ein Rückschluss auf die beste Wahl der Alternativen gezogen und so eine durchdachte rationale Entscheidung getroffen. Die Entscheidung wird dann durch die Aktuatoren der Dual Reality in der Wirklichkeit nachgebildet. Dieses Verfahren beschreibt Echtzeitentscheidungen durch Trajektorensimulation in Dual Reality.

1.2 Zielsetzung

Die Arbeit konkretisiert fünf Ziele. Das wichtigste Ziel ist die Entwicklung eines Rahmenwerkes für Trajektoriensimulation, um Echtzeitentscheidungen in Dual Reality durchführen zu können. Um dieses zu testen muss ein geeignetes Testszenario konzipiert und umgesetzt werden. Anschließend soll das ursprüngliche Ziel dieser Arbeit, ein Kundenleitsystem für große Supermärkte prototypisch umgesetzt werden. Damit das Rahmenwerk überhaupt entwickelt werden kann, wird ein mathematisches Verständnis der Trajektoriensimulation benötigt und daher soll diese mathematisch definiert werden. Optional wird noch ein Assistenzsystem umgesetzt, welches Filialleiter beim Öffnen und Schließen von Kassen unterstützen soll. Die genauen Ziele spezifizieren sich also wie folgt:

- **Die Entwicklung eines Rahmenwerks**

  Um das Verfahren der Entscheidungsfindung zu vereinfachen, soll ein Rahmenwerk erstellt werden, welches die Echtzeitentscheidungen durchführt. Dieses Rahmenwerk soll die Entwicklung der zur Entscheidung benötigten Komponenten anleiten und so vereinfachen. Damit das Rahmenwerk eine Entscheidung fällen kann, muss der Anwender Simulatoren, Bewertungsfunktionen und die Alternativen spezifizieren. Wie viele Trajektorien erstellt und ausgewertet werden sollen, wird über Parameter festgelegt. Das Rahmenwerk soll für Mehrkernprozessoren ausgelegt sein und die Erstellung sowie die Auswertung der Trajektorien parallel durchführen. Der Name des Rahmenwerks ist „Trajectory Simulation Decision Framework“ und es wird in dieser Arbeit kurz TSDF genannt.

- **Die Entwicklung eines Testszenarios**

  Das TSDF soll an einem erfolgversprechenden Szenario getestet werden. Die Steuerung einer Ampel wäre ein solches Szenario. Aktuell werden Ampeln hauptsächlich von Zeitschaltungen gesteuert. Da die Zeitschaltungen nicht dynamisch auf den Verkehr reagieren, besteht hier ein großes Verbesserungspotential. Zum Testen des Rahmenwerks soll deshalb eine Kreuzung mit Sensoren instrumentalisiert werden und die Ampeln steuerbar gemacht werden. Dadurch wird die Ampel in eine Dual Reality überführt und bietet eine geeignete Grundlage für das TSDF. Dieses Szenario wird fortan als eTAS „electronical Traffic-Light Arbitration Service“ bezeichnet.

- **Die Entwicklung eines Kundenleitsystems**

  Das TSDF ist mit dem Ziel, ein Kundenleitsystem für die Globus GmbH zu entwickeln, entstanden und daher ist das Kundenleitsystem ein elementarer Bestandteil dieser Arbeit. In großen Supermärkten kann die Entscheidung der Kassenwahl

nicht rational getroffen werden, da durch die Vielzahl der Kassen ein Überblick nicht möglich ist. Dies führt, laut Globus GmbH, zu einer ungleichen Auslastung der Kassen. Diesem nicht erwünschten Verhalten soll ein Kundenleitsystem entgegen wirken. Dazu sollen Monitore die Kunden bei der Kassenauswahl unterstützen. Dies soll einen weiteren Anwendungsfall des TSDF darstellen. Obwohl hier die Entscheidung letztendlich durch den Kunden gefällt wird, muss das TSDF dennoch autonom entscheiden, welche Kasse auf welchem Monitor beworben wird. Das TSDF löst in diesem Fall also nicht die Frage, welche Kasse zu wählen ist, sondern fällt die Entscheidung, was geeigneter Weise auf einem Monitor in der aktuellen Situation anzuzeigen ist. Die Aussichten auf einen Erfolg sind jedoch weitaus geringer als beim eTAS, da berücksichtigt werden muss, dass manche Kunden die Monitore ignorieren. Wie groß die Auswirkung der Monitore auf die Kunden ist und wo die Monitore geeigneter Weise aufgestellt werden, soll in dieser Arbeit ermittelt werden. Als eine Alternative zu den Monitoren wird optional das Leitsystem auf eine Beratung über mobile Endgeräte des Kunden erweitert. Auf Anfrage fällt das System für den Kunden die Entscheidung, welche Kasse er wählen soll und teilt diesem die berechnete beste Alternative mit. Das Kundenleitsystem wird im folgenden als eCASE „electronical Cashzone Advisory ServicE“ bezeichnet.

- **Die Entwicklung eines Kassenöffnungssystems**

  Um die Flexibilität dieses Ansatzes zu unterstreichen, soll optional mit der Hilfe des gleichen Modells ein Beratungssystem für Marktmanager entwickelt werden, das autonom die Entscheidung fällt, wann eine Kasse geöffnet oder geschlossen wird. Dabei soll nicht nur die Entscheidung, ob eine Kasse geöffnet oder geschlossen wird, gefällt werden, sondern auch die Entscheidung welche Kasse geöffnet oder geschlossen wird. Dieses System wird im folgenden als eCOS „electronical Caszone Opening Service“ bezeichnet.

- **Die mathematische Fundierung.**

  Die Entscheidungsfindung über Trajektorensimulationen soll mathematisch definiert werden, um so einen Grundstein für die dazugehörige Theorie zu legen. Dies soll helfen die Entscheidungsfindung besser zu verstehen, um so die Wahl der Parameter besser treffen zu können. Es soll gezeigt werden, dass durch eine Erhöhung der Anzahl der Simulationen die Qualität der Entscheidung gegen das Optimum konvergiert. Diese Konvergenz ist wichtig, da es zwar theoretisch möglich ist, unendlich viele Simulationen durchzuführen, praktisch können jedoch nur wenige Simulationen wegen beschränkter Ressourcen durchgeführt werden. Die Frage ist, ob eine konsequente Erhöhung der Ressourcen automatisch zu einer Verbesserung der Qualität der Entscheidungen führt. Da für die Anfertigung der Arbeit nur geringe Hardwareressourcen (zum Beispiel keine Cloud) zur Verfügung stehen, soll der theoretische Nachweis genügen, dass mehr Simulationen zu besseren Ergebnissen führen.

1.3 Abgrenzung

Da eine Masterarbeit einen zeitlichen Rahmen hat und einige wünschenswerte Ziele daher nicht umsetzbar sind, soll im folgenden erklärt werden, welches diese Ziele sind und warum sie nicht durchgeführt werden können:

In dieser Arbeit kann das naheliegende Ziel, die Prozesse der Entscheidungsfindung an realen Umgebungen zu testen, nicht behandelt werden, da hierfür entsprechende Ressourcen nicht bereit gestellt werden können. Die Instrumentalisierung einer echten Ampel sowie die vollständige Instrumentalisierung eines Supermarktes wäre ein kostspieliger Aufwand und sollte erst nach den positiven Resultaten dieser Arbeit in Angriff genommen werden. Daher werden in dieser Arbeit die realen Umgebungen ebenfalls simuliert. Diese *Simulierten Realitäten* werden mit künstlichen Sensoren und Aktuatoren ausgestattet und bieten so eine geeignete Methode, die hier vorgestellten Techniken zu testen. Durch eine Trennung der *Simulierten Realität* durch eine Netzwerkschicht ist die Anbindung an eine echte Realität später einfacher und es sichert die Authentizität der Dual Reality, da lediglich Sensordaten und Aktuatorenbefehle übertragen werden.

Der eTAS soll keine Konkurrenz zu bestehenden Ampelsteuerungssystemen bilden und wird lediglich als Testszenario verwendet. Auf einen Vergleich mit anderen Systemen, wie die zum Beispiel in SUMO [25] implementierten, soll daher verzichtet werden. Die exakte und verkehrsgerechte Umsetzung, die unter anderem im Handbuch für Verkehrskontrollsysteme beschrieben ist [13], muss bei einer Umsetzung in der Realität berücksichtigt werden, kann in dieser Arbeit aber vernachlässigt werden.

Da der ursprüngliche Fokus dieser Arbeit die Entwicklung eines Kundenleitsystems für Globus ist, kann kein direkter Vergleich mit anderen KI-Systemen gezogen werden. Die Implementierung des Szenarios in mehreren KI-Algorithmen übersteigt die zur Verfügung stehende Zeit, weshalb dieser Vergleich nur auf einer theoretischen Ebene durchgeführt wird. Dennoch wird bewusst festgestellt, dass dieser direkte Vergleich eine wichtige zukünftige Aufgabe sein muss, um den hier vorgestellten Ansatz der Trajektoriensimulation geeignet mit anderen KI-Methoden zu vergleichen.

Die mathematische Definition eröffnet Möglichkeiten die Fehlereinschätzung zu untersuchen. Zusätzlich könnten Beweise der Konvergenz sowie weitere Theorien der Statistik angewandt werden, um weitere Informationen über Trajektoriensimulation zu bekommen. Da das Hauptziel die Umsetzung eines Rahmenwerkes ist, soll die Theorie jedoch nur definiert werden und so das Verständnis verbessern.

1.4 Gliederung

In Kapitel 2 werden verwandte Techniken vorgestellt, die zur Entscheidungsfindung verwendet werden. Da in der Verkehrsführung bereits intelligente Systeme eingesetzt werden, sollen diese kurz beschrieben werden. Zusätzlich wird die Fähigkeit der menschlichen Entscheidungsfindung eingegrenzt und mit der Trajektoriensimulation verglichen. Kapitel 3 stellt zuerst das Konzept der Trajektoriensimulation exemplarisch vor und veranschaulicht den Vergleich zu einer menschlichen Testgruppe. Anschließend wird die Trajektoriensimulation mathematisch definiert und dabei die entscheidende „k-besser"-Relation zur Bewertung der Alternativen vorgestellt und erklärt. Mit Hilfe der mathematischen Definitionen wird dann ein Rahmenwerk sowie drei Anwendungsfälle dieses Rahmenwerks konzipiert: Die Steuerung einer Ampelkreuzung, ein Kundenleitsystem für große Supermärkte und ein Entscheidungsunterstützungssystem für Filialleiter beim Öffnen und Schließen von Kassen. In Kapitel 4 wird die vorgestellte Trajektoriensimulation mit anderen KI-Methoden verglichen. Dazu werden die KI-Systeme geprüft, ob sie geeignet sind Entscheidungen durchzuführen, ob sie parallelisierbar sind, ob durch diese Parallelisierung die Qualität der Entscheidungen steigt und ob die Laufzeit vorhersagbar ist. Am Ende des Kapitels werden die Ergebnisse tabellarisch zusammengefasst. Das Kapitel 5 stellt die Implementierung der Komponenten des Rahmenwerks vor. Anschließend werden die Implementierungen der drei Anwendungsfälle grob erläutert. Im Kapitel 6 werden die Ergebnisse der Anwendungsfälle sowie die Laufzeitmessungen gezeigt und analysiert. Kapitel 7 fasst die Arbeit zusammen und gleicht die vorgestellten Ziele mit den erreichten Zielen ab. Im Appendix finden sich weitere Grafiken sowie ein Leitfaden zur Erstellung einer Dual Reality.

2 Verwandte Arbeiten

In diesem Kapitel sollen Systeme und Arbeiten vorgestellt werden, die sich mit der Entscheidungsfindung beschäftigen. Da die Verkehrssteuerung ein wesentlicher Bestandteil ist, werden kurz auch andere Verkehrskontrollsysteme vorgestellt. Am Ende dieses Kapitels werden wichtige Erkenntnisse aus der menschlichen Entscheidungsfindung zusammen gefasst und auf die Trajektoriensimulation bezogen.

2.1 Entscheidungsunterstützungssysteme

Entscheidungsunterstützungssysteme kurz DSS (engl. Decision Support Systems) sind Programme, die Menschen bei der Wahl einer Entscheidung durch das Bereitstellen von Informationen unterstützen. Sie gehen auf die Veröffentlichung eines Rahmenwerks für Management Informationssysteme von Gorry und Scott Morton zurück [14]. Das Interesse an Entscheidungsunterstützungssystemen spiegelt sich in der Teilnehmerzahl beim Tutorial über „DSS in Forschung und Praxis“ auf der ersten spezialisierten IFORS Konferenz über Decision Support Systeme wieder. An diesem Tutorial nahmen von 300 Konferenzteilnehmern etwa die Hälfte teil [2]. DSS finden ihre Anwendung unter anderem in der Landwirtschaft, Medizin, Luftfahrt und im Management [1]. Wie der Name schon sagt, unterstützen sie lediglich bei der Entscheidungsfindung. Sie gliedern sich nach der Art der Informationsbereitstellung [1, 15, 10]:

- **Passive DSS** helfen lediglich im Prozess der Entscheidungsfindung durch das Bereitstellen von Informationen.
- **Aktive DSS** liefern fertige Lösungen zusammen mit den Informationen, überlassen die Wahl jedoch einem Menschen.
- **Kooperative DSS** liefern fertige Lösungen und Informationen, ein Mensch kann diese jedoch vor der Wahl anpassen und modifizieren.

Zusätzlich unterscheidet man noch nach der Art, wie die Informationen gewonnen werden. Besonders relevant für diese Arbeit sind dabei die modellgetriebenen Entscheidungsunterstützungssysteme (vgl. Dicodess [10]), da sie Modelle zur Ableitung der Zukunft benutzen, um die Informationen zu generieren. Neben ihnen gibt es noch kommunikationsgetriebene, datengetriebene, dokumentgetriebene und wissensgetriebene Entscheidungsunterstützungssysteme.

Trotz dem gleichen Grundgedanken, Modelle und Simulationen für Entscheidungen heran zu ziehen, werden die Entscheidungen nicht automatisiert getroffen, sondern Menschen überlassen. Ebenfalls werden zur Simulation meistens sehr abstrakte Modelle herangezogen. Die Modelle in dieser Arbeit sind dagegen detaillierter und die Entscheidungen werden völlig autonom vom System getroffen und umgesetzt.

2.2 Echtzeitentscheidungsunterstützungssysteme

Echtzeitentscheidungsunterstützungssysteme (kurz EEUS) sind spezielle Entscheidungsunterstützungssysteme, die für schnelle spontane Entscheidungen optimiert sind. Sie kommen vor allem im Finanzsektor vor [53], werden aber auch im Data-Warehousing [41] und in der Robotik verwendet [20].

In der Entscheidungstheorie unterscheidet man „Programmed Decisions“ und „Non Programmed Decisions“ [50]. Bei „Programmed Decisions“ könnte man im Deutschen von alltäglichen Entscheidungen sprechen. Es handelt sich um Entscheidungen die oft wiederkehren und gut verstanden sind. Ein Beispiel dafür wäre ein Postbote, der Briefe in Kästen einsortiert. Die Entscheidung, in welches Fach ein Brief gehört, wäre eine solche „Programmed Decision“. Die „Non Programmed Decisions“ sind dagegen plötzlich auftretende Entscheidungen. Der wesentliche Hintergrund dieser Unterscheidung ist, dass EEUS nur für „Programmed Decisions“ entwickelt werden können. Ohne den wiederkehrenden Charakter könnte keine Anwendung für diesen Fall geschrieben werden. Diese Einschränkung bezieht sich auch auf das in dieser Arbeit vorgestellte System. Die Entscheidungen müssen zur Entwicklungszeit spezifiziert werden und daher ist es nicht möglich, spontan auftretende unbekannte Entscheidungen durch das System fällen zu lassen.

Im Finanzsektor helfen diese Echtzeitentscheidungssysteme beim Zusammenstellen von Portfolios [53]. In der Arbeit „Real Time Decision Support for Portfolio Management“ stellen die Autoren ein System vor, welches „Object Oriented Bayesian Knowledge Bases“ verwendet, um die von einem Finanzexperten vorgeschlagene Portfolios zu überprüfen und dem Finanzexperten schnell Informationen über die möglichen Gewinne zu liefern. Das System braucht für eine solche Einschätzung gerade mal 2.03 Sekunden, sofern das detaillierte Model zugrunde gelegt wird. Nutzt man dagegen ein abstraktes Model, so können die Einschätzungen bereits in 0.26 Sekunden bereit gestellt werden. Dies ermöglicht Finanzmanagern kritische Entscheidungen schnell zu überprüfen und somit in Echtzeit auf Kursschwankungen reagieren.

akeshi Fukase et al. [9] stellen in ihrem Artikel über „Real Time Decision Making under Uncertainty of Self-localization Results“ vor, wie ein Roboter mit ungewissen Informationen über die eigene Position dennoch schnell gute Entscheidungen treffen kann. Sie nutzen dafür Dynamische Programmierung und vergleichen zwei verschiedene Implementierungen. Das Projekt wurde im Rahmen des Robocup[20] entwickelt. Die Roboter verfügen nur über einen geringen Speicher und ungenaue Sensoren. Die Aktuatoren zur Bewegung sind ebenfalls unpräzise und so kann die Position nicht exakt bestimmt werden. Wegen des geringen Speichers werden zur Positionierung „Vector Quantizations“ als Kartenmaterial herangezogen. Trotz dieser massiven Einschränkungen soll der Roboter in der Lage sein, einen beliebigen Zielzustand aus jedem aktuellen Zustand erreichen zu können. Dabei gehen die Autoren von folgenden Annahmen aus:

- „Es gibt acht diskrete Bewegungsaktionen und eine Beobachtungsaktion“
- „Die Bewegungsaktionen unterliegen starken Fehlern“
- „Es gibt sechs Orientierungspunkte am Rand des Feldes“
- „Die Abstandsmessungen zu den Orientierungspunkten sind stark fehlerbehaftet“

- „Der Roboter sieht nur auf den Ball, wenn er auf ihn zuläuft“
- „Der Roboter schwingt seinen Kopf horizontal zur Positionsbestimmung während der Beobachtungsaktion“

Die beiden Implementierungen unterscheiden sich grundlegend. Das eine System verwendet Karten um die Position zu bestimmen, die andere Implementierung dagegen Schwellenwerte. Die Positionierung über die Schwellenwerte erreicht dabei eine Erfolgsrate von 90%, wenn es darum geht einen beliebigen Zielzustand zu erreichen. Dazu wurden drei Szenarien mit jeweils 10 Testdurchläufen getestet. Im ersten Szenario konnten 8 Testdurchläufe erfolgreich absolviert werden, im zweiten Szenario 9 und im dritten Szenario 10. Bei der Verwendung von Kartenmaterial konnte eine Erfolgsrate von 100% erreicht werden.

Zentral bei dieser Arbeit ist auch die Geschwindigkeit. Der Roboter muss Entscheidungen über seinen Zustand in Echtzeit treffen, um den Übergang in einen Zielzustand berechnen zu können.

2.3 Monte-Carlo-Baumsuche

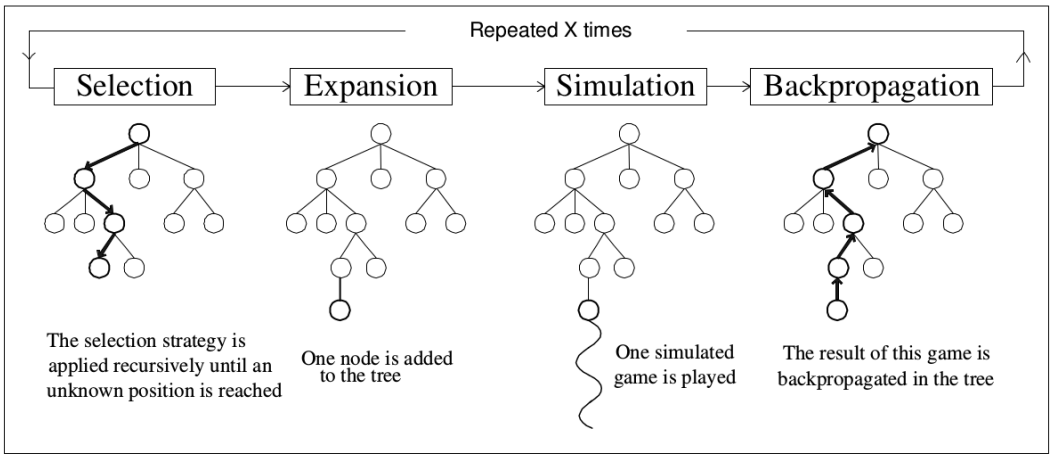


Abbildung 2.1: Monte Carlo Simulationsbaum [6]

Bei Monte-Carlo-Baumsuche (MCTS engl. Monte Carlo Tree Search) wird eine Bestensuche verwendet, welche anstatt von einer Heuristik durch Monte Carlo Simulationen geleitet wird. MCTS sind besonders geeignet für Probleme, bei denen eine komplette Baumsuche zu zeitaufwendig ist [6]. Besondere Erfolge erzielen die MCTSs bei Computerspielen. Sie werden erfolgreich für rundenbasierte Spiele wie Go [4, 6, 58], Backgammon [54] und Schach [37] eingesetzt. Es gibt aber auch erste Versuche mit „Real-Time-Strategy-Games“ wie „Age of Empires“ [8].

Chaslot beschäftigt sich in seiner Doktorarbeit [6] intensiv mit dem Thema Monte Carlo Simulationsbäume. Mit der Grafik 2.1 zeigt er die Schritte, die bei Monte Carlo Simulationssuchbäumen mehrmals durchlaufen werden müssen, bevor eine Entscheidung getroffen werden kann:

1. Die Selektionsstrategie wird über den bereits teilweise explorierten Baum iteriert, bis man bei einem zufälligen Knoten angekommen ist. Dieser Knoten muss nicht zwingend ein Blattknoten des explorierten Baums sein.

2. Durch die Expansion wird zu diesem Knoten ein weiterer möglicher Kindknoten hinzugefügt, in dem man im aktuellen Knoten eine bisher nicht durchgeführte Aktion anwendet. Geht man zum Beispiel davon aus, dass jeder Knoten den Zustand eines Schachbrettes darstellt, so wird ein Kindknoten erzeugt, indem man einen gültigen Zug durchführt, der noch nicht exploriert wurde.

3. Bei der Simulation wird von diesem neuen Knoten aus ein komplettes Spiel zufällig bis zum Ende durchgeführt. Alternativ kann eine vorhandene bekannte KI genutzt werden, um dieses Spiel zu simulieren. Wurde das Spiel gewonnen oder verloren, so gibt dies statistisch Rückschlüsse darauf, wie gut der durchgeführte Zug war.

4. Der Ausgang der Simulation wird im letzten Schritt dazu benutzt, um die Werte aller Knoten bis zum Wurzelknoten entsprechend anzupassen. Erfasst werden Minima, Maxima und Durchschnitt.

Der so gewonnene Baum wird für die Entscheidungsfindung benutzt. Dabei wird nach den X Durchläufen die erste Ebene nach der Wurzel ausgewertet. Man kann sich anhand der dort vorhandenen Minima, Maxima und Durchschnittswerte für einen Spielschritt entscheiden. Dieser ist dann zumindest statistisch optimal. Für $X \to \infty$ ergibt sich nach dem Gesetz der großen Zahlen die komplette Exploration des Baums und die Werte konvergieren gegen die eines Min-Max-Baums. Der Vorteil gegenüber dem Min-Max-Baum ist, dass weniger Explorationen benötigt werden. Dadurch können auch Bäume mit sehr weiter Ausbreitung, wie beim Spiel Go, erfolgreich verwendet werden.

Der vorgestellte Algorithmus für die MCTS funktioniert jedoch nur bei Spielen mit diskreten Zustandstandsräumen, da sonst bereits Schwierigkeiten bei der Selektion auftreten. Gibt es zum Beispiel auf jeder Ebene unendlich viele Möglichkeiten einen Zug durchzuführen, würde bei der Selektion jedes mal ein komplett unbekannter Knoten gewählt und exploriert werden. Dies würde dazu führen, dass es immer nur eine *Backpropagation* gäbe und somit nur einen ermittelten Zahlenwert in der ersten Ebene. Dadurch würde die statistische Relevanz der Bewertung dieses Zugs gegen Null konvergieren.

2.4 Verkehrskontrollsysteme

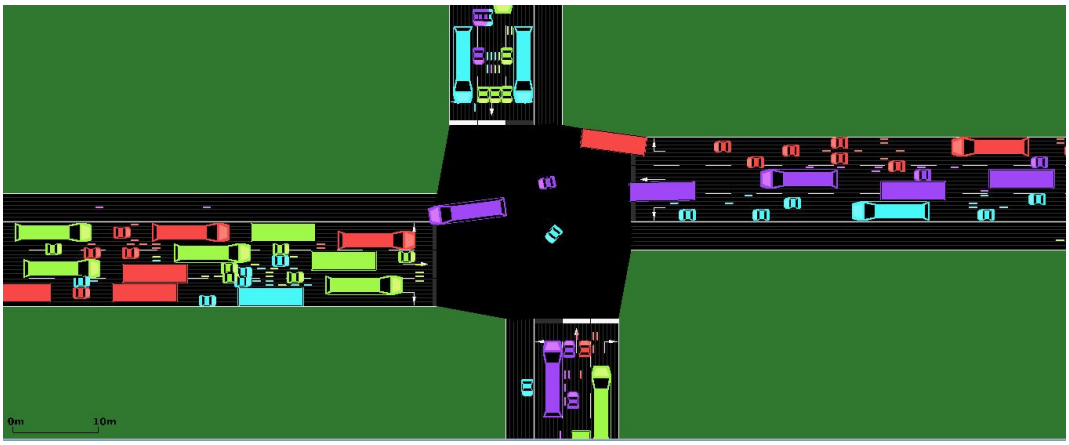


Abbildung 2.2: SUMO - Simulation of Urban MObility
`http://www.sumo-sim.org/`

Das SUMO Projekt wurde 2001 als quelloffenes Softwareprojekt gestartet. In erster Linie sollte es die Vielzahl von Verkehrskontrollsystemen unter einem vergleichbaren Standard vereinen [39]. SUMO ist in der Lage große Verkehrsnetze zu simulieren. Dazu werden diese über XML-Dateien spezifiziert und anschließend von einem Server ausgeführt. Dieser Server teilt den über ein Netzwerk verbundenen Clients Daten der platzierten Sensoren mit und ermöglicht die Steuerung der Ampeln. Zusätzlich können spezielle Situationen wie Unfälle oder Baustellen leicht umgesetzt werden. SUMO benutzt ein mikroskopisches Modell, welches von Stefan Krauss am deutschen Zentrum für Luft und Raumfahrt [25] entwickelt wurde.

Trotz zahlreicher Publikationen gibt es laut Roozemond [46] nur wenig implementierte fertige Lösungen zur Verkehrssteuerung, wie zum Beispiel UTOPIA-SPOT oder SCOOT. Zwei weitere Systeme, die dazu in der Lage sind sich selbst zu optimieren, MOVA (Vereiniges Königreich) und LHOVRA (Schweden), werden bereits auf isolierten Kreuzungen eingesetzt.

Um Entscheidungssysteme besser entwerfen zu können, gibt es ein Handbuch [13], welches aktuelle Entwicklungen und Standards erklärt und definiert. Der Forschungsbericht enthält alle Details, die man über Verkehr und Verkehrsfluss wissen muss: von üblichen Variablen, bis hin zu detaillierten Beschreibungen der Ampelphasen und üblichen Spurlängen. Zusätzlich beschreibt es verwendete Sensoren und die Protokolle der üblichen Kontrollgeräte.

Zur Simulation von Verkehrssystemen und für die Entscheidungen von Ampelsystemen gibt es verschiedene Ansätze: Fuzzy Logik [22], Agentensysteme [46], Neuronale Netze, Genetische Algorithmen, Markov Entscheidungsprozesse und Unterstützendes Lernen [39, 23].

Robinson-Mosher und Christopher Egner versuchen in ihrer Arbeit Regeln zur Steuerung von Ampeln zu lernen. Dabei werden Simulationen verwendet, die mit der Hilfe von Markov-Entscheidungsprozessen analysiert werden [45]. Das Lernen der Regeln ist laut den Autoren nicht so erfolgreich, wie sie ursprünglich angenommen haben. Sie vermuten, dass die verwendeten Zufallsvariablen zu viele Informationen der Kreuzung

abstrahieren. Die abstrakte Darstellung der Kreuzung über Zufallsvariablen unterliegt also dem mikroskopischen Ansatz von SUMO.

Aufgrund der ermittelten Informationen wird in dieser Arbeit auch ein mikroskopischer Ansatz verfolgt, der die Autos und die Kreuzung als objektorientiertes Klassenkonstrukt darstellt und die Bewegung der Autos bis auf eine automatische Bestimmung der Beschleunigung reduziert. Da das Verkehrskontrollsystem lediglich ein Testszenario ist, wird SUMO nicht verwendet, da sehr viel mehr Details in die Virtuelle Realität übernommen werden müssten. Außerdem wird für die Duale Realität ebenfalls ein Simulator benötigt, der möglichst identisch zu dem in SUMO sein müsste. Da dieser sowieso entwickelt werden müsste, stellt die Verwendung von SUMO höchstens einen erheblichen Wert für die Vergleichbarkeit der Zahlen dar, würde jedoch die Entwicklungszeit erhöhen. Nachdem allerdings die Ergebnisse für die Verkehrssteuerung unter der Verwendung von Trajektoriensimulation sehr gut sind, sollte die Implementierung für SUMO erneut abgewägt werden, um sie mit dem Stand der Technik anderer Systeme vergleichen zu können. SUMO scheint außerdem nicht in der Lage zu sein, im Zeitraffer zu simulieren, und so würde eine Bewertung und das Testen der Arbeit schwerer fallen.

2.5 Menschliche Entscheidungsfindung

Als Vorbild für künstliche Systeme wählt man oft die Natur. Es stellt sich jedoch die Frage, wie gut Computer gegenüber ihren Vorbildern sind. Laut Bruner et al. [5] haben die Menschen zum Beispiel Schwierigkeiten mit der Informationsverarbeitung. Sie lassen sich leicht ablenken und Entscheidungen werden so sehr oft eher intuitiv und nicht rational gefällt. Die Psychologie hat auch bereits herausgefunden, dass Menschen Informationen im Wesentlichen nur sequentiell [49] verarbeiten können. Hier sind sie gegenüber einem Computer natürlich im Nachteil. Computer können durch die wachsende Anzahl an Prozessoren immer besser parallel Informationen verarbeiten. Betrachtet man nun die Forschungen von Miller at al, so erfahren wir, dass Menschen sogar bei der sequentiellen Informationsverarbeitung stark eingeschränkt sind. Es wurde festgestellt, dass Menschen nicht mehr als 9 verschiedene Situationen durchdenken können [38]. Waugh und Normal konnten dies in Umgebungen mit starker Ablenkung auf maximal 2 Situationen einschränken, die ein Mensch durchdenken kann [57]. Dies ist ein gutes Indiz dafür, dass Hilfssysteme die Entscheidungen von Menschen verbessern können. Ferner lässt die geringe Anzahl von Situationen, die ein Mensch normalerweise durchdenkt um seine Entscheidungen zu treffen, annehmen, dass ein Computer, der wesentlich mehr Situationen in gleicher Zeit durchdenken kann, bessere Entscheidungen trifft. Menschen durchdenken allerdings keine komplett zufälligen Situationen, sondern tun dies nach einer bisher unverstandenen Systematik.

Im Gegensatz zum Menschen lassen sich Computer nicht ablenken. Allerdings ist die Wahrnehmung über Sensoren qualitativ noch nicht mit der menschlichen Wahrnehmung vergleichbar. Quantitativ jedoch ist der Computer überlegen. Er kann Daten aus mehr Sensoren verarbeiten, die einen größeren Raum abdecken.

Nach Morrin [3] kann die menschliche Entscheidungsfindung verbessert werden, wenn man unnütze Informationen ausblendet. Im Kassenszenario dieser Arbeit würde dies bedeuten, dass ein Monitor, der eine gute Kasse empfiehlt, die Entscheidungsfindung verbessert, da der Mensch sich auf die Entscheidung des Computers konzentriert und diese lediglich verifizieren muss.

3 Konzept

Wie bereits in der Einleitung erklärt, benötigt man, um eine rationale Entscheidung zwischen mehreren Alternativen zu fällen, ein konkretes Verständnis der Gegenwart. Man muss außerdem in der Lage sein, die Zukunft der verschiedenen Alternativen ableiten zu können. Eine Alternative ist eine Veränderung, die zu einem Zeitpunkt x in der Zukunft eintritt. Zur Bewertung muss allerdings ein noch weiter in der Zukunft liegender Zeitpunkt y abgeleitet werden. Abbildung 3.1 zeigt die Entscheidung für den Aktuatorzustand A, der vom System evaluiert werden soll und entweder 0 oder 1 sein kann. Dazu werden beide Alternativen jeweils zweimal evaluiert. Dabei gilt n ist die Anzahl der Alternativen (hier $n = 2$) und k ist die Anzahl der Simulationen (hier $k = 2$). Ausgehend von der Gegenwart, werden mit Hilfe eines Simulators x zufällige Zeitschritte der Zukunft berechnet. Bei zwei der vier Simulationen wird A auf den Wert 0 festgelegt und bei den anderen zwei wird A auf den Wert 1 festgelegt. Anschließend wird jeweils bis zum Zeitpunkt y weiter simuliert. Dadurch entstehen $n \cdot k = 4$ Trajektorien, jeweils $k = 2$ für jeden Zustand.

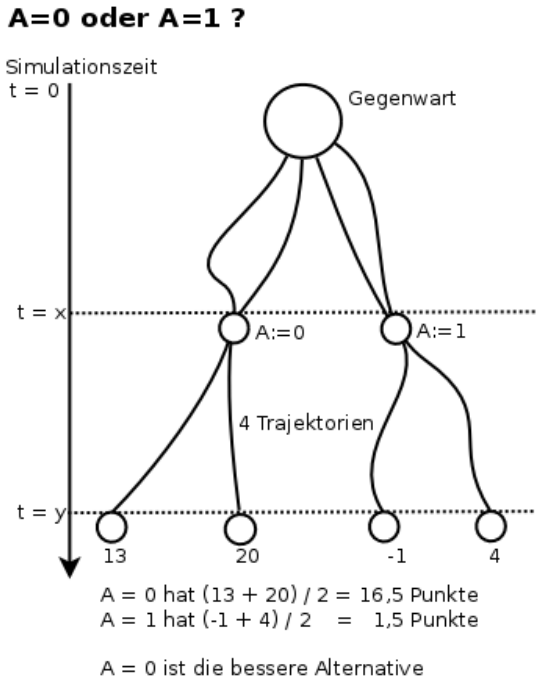


Abbildung 3.1: Konzept der Trajektoriensimulation

Die Bewertung dieser Trajektorien ergibt Beispielsweise 13, 20, -1 und 4. Die Berechnung des Durchschnitts für jede Alternative liefert eine Einschätzung des Erwartungswerts der Bewertungsfunktion f für die Alternativen $E(f(A = 0)) \overset{2}{\approx} \frac{13+20}{2} = 16.5$ und $E(f(A = 1)) \overset{2}{\approx} \frac{-1+4}{2} = 1.5$. Dies lässt den statistischen Schluss zu, dass der Erwartungswert $E(f(A = 0)) > E(f(A = 1))$. Der Computer entscheidet in diesem Fall A zum Zeitpunkt x auf 0 zu setzen.

Mathematisch betrachtet ist die Trajektoriensimulation also nichts anderes als die Approximation des Erwartungswertes einer Bewertungsfunktion f. Es gilt also:

$$\lim_{k\to\infty} \sum_{i=0}^{k} \frac{f(A = a)}{k} = E(f(A = a))$$

Damit ist gezeigt, dass mit steigender Anzahl k der Erwartungswert genauer bestimmt wird und somit die Qualität der Entscheidung, die auf einem Vergleich der Erwartungswerte basiert, verbessert wird.

Im folgenden wird ein kleines Beispiel betrachtet, um zu zeigen, wie diese mathematische Grundlage verwendet werden kann:

Beispiel 1 (Münzwurfspiel)**.** *Gegeben sei ein Spiel, bei dem eine Münze geworfen wird. Der Spieler muss sich vor dem Beginn für Kopf oder Zahl entscheiden. Hat der Spieler sich für Kopf entschieden und er gewinnt den Münzwurf, erhält er 20 Punkte. Verliert er bei Kopf, bekommt er 10 Punkte abgezogen. Hat er sich dagegen für Zahl entschieden und gewinnt den Münzwurf, erhält er 30 Punkte. Verliert er bei Zahl, bekommt er 25 Punkte abgezogen.*

Ein Mensch, der sich mit Wahrscheinlichkeitsrechnung auskennt wird direkt erkennen, dass man bei Kopf einen Erwartungswert von $(20 - 10)/2 = 5$ und bei Zahl einen Erwartungswert von $(30 - 25)/2 = 2.5$ erhält. Sollen mehrere Spieldurchläufe gespielt werden, würde er sich natürlich für Kopf entscheiden.

Zum Vergleich wurde dieses Spiel mit 20 Drittklässlern für eine Schokoladenbelohnung in einer gestörten Umgebung durchgeführt [1]. Die gestörte Umgebung wurde gewählt, da hier, laut Waugh [57], nur zwei Varianten durchdacht werden können, das Spiel jedoch vier mögliche Varianten hat. Drittklässler wurden gewählt, da sie kein Wissen über Wahrscheinlichkeit verfügen und daher die Entscheidung nur durch Durchdenken lösen können. 85% (17 von 20) der Drittklässler entschieden sich für Zahl, da sie vermutlich den höheren Gewinn als Aussicht sahen und das Risiko falsch abschätzten. Dennoch würde der Computer bereits mit einer Simulation pro Alternative, also beim Durchdenken von insgesamt zwei Varianten (wie auch die Kinder), ein besseres Endergebnis erreichen. Denn die möglichen Ausgänge der Simulation wären:

- Der Computer gewinnt in beiden Simulationen mit 20 bzw. 30 Punkten. Der Computer würde sich deshalb falsch entscheiden, da $30 > 20$.
- Der Computer gewinnt in der ersten Simulation und verliert in der zweiten Simulation mit 20 bzw. -25 Punkten und würde sich daher richtig entscheiden, da $20 > -25$.
- Der Computer verliert in der ersten Simulation und gewinnt in der zweiten Simulation mit -10 bzw. 30 Punkten und würde sich daher falsch entscheiden, da $30 > -10$.
- Der Computer verliert in beiden Simulationen mit -10 bzw. -25 Punkten und würde sich richtig entscheiden, da $-10 > -25$.

Der Computer würde sich also in 50% der Fälle richtig entscheiden und deshalb einen Gewinn von etwa $0.5 \cdot 5 + 0.5 \cdot 2.5 = 3.75$ Punkten pro Spiel erwirtschaften[2]. Die Drittklässler haben sich in 85% der Fälle falsch entschieden, was $0.15 \cdot 5 + 0.85 \cdot 2.5 = 2.875$ Punkte pro Spiel bedeuten würde. Statistisch wird der Computer also in vielen Spieldurchläufen gegen die Grundschüler gewinnen und dies bereits bei einer Simulation pro Alternative.

Um die Simulationen jedoch überhaupt zu ermöglichen und somit Entscheidungen zu fällen, wird die Gegenwart benötigt, die bisher als gegeben angenommen wurde. Ebenso

[1] Die gestörte Umgebung wurde durch einen gezeigten Film im Hintergrund realisiert

[2] Dabei sind 2.5 und 3.75 die bereits berechneten Erwartungswerte.

müssen für Echtzeitentscheidungen in einer instrumentalisierten Umgebung Aktuatoren vorhanden sein, die eine solche Entscheidung umsetzen. Da die Aktuatoren eine gewisse Zeit zur Umsetzung brauchen, wird die Änderung in der Simulation auch erst zu einem Zeitpunkt x durchgeführt. Da die Gegenwart nur über Sensoren wahrgenommen wird und Sensoren fehlerbehaftet sind, muss ferner angenommen werden, dass der berechnete Erwartungswert ebenfalls fehlerbehaftet ist. Ist die Abweichung zur Realität zu groß, kann eine sinnvolle Entscheidungsfindung nicht mehr garantiert werden. Vergleichbar wäre es, wenn ein Mensch zum Beispiel eine Ampel steuert, jedoch keine Informationen über den Verkehr bekommt. Intuitiv würde er vielleicht jede Phase gleich lang einstellen, ohne zu wissen, dass vielleicht 95% der Autos aus einer Richtung kommen. Seine Entscheidungsfindung wäre zwar berechtigt, dennoch würde seine Entscheidung vermutlich zu einem Chaos auf der Kreuzung führen. Je genauer die Informationen jedoch auf die 95% hindeuten, desto mehr würde er die Zeit auf die entsprechende Phase verschieben und so das Chaos verringern oder gar komplett verhindern. Würde man dagegen einen Menschen wählen, der die exakten Daten über den Verkehrsfluss bekommt, aber keine Ahnung von Verkehr und Autos hat, so würde seine Entscheidung auch zu einem Chaos führen.

Das Selbe muss für Trajektoriensimulationen angenommen werden. Je genauer die Informationen für die Simulation sind, desto besser sind die Entscheidungen. Wird jedoch der Verkehrsfluss falsch simuliert, so führen auch präzise Informationen nicht zu einer guten Entscheidung. Der Übergang zwischen guten und schlechten Entscheidungen ist fließend. Daher kann die Trajektoriensimulation auch mit nur realitätsnahen Informationen und fehlerbehafteten Sensordaten noch gute Entscheidungen fällen.

3.1 Mathematische Definition

Nachdem exemplarisch gezeigt wurde, dass eine Entscheidung durch Trajektoriensimulation eine Approximation des Erwartungswerts einer Bewertungsfunktion ist, soll dies im folgenden dieser Arbeit mathematisch genauer definiert werden. Die Definitionen des Phasenraums und der Trajektorien orientiert sich an ihren mathematischen Vorbildern. Die exakten Definitionen wurden im Rahmen dieser Arbeit neu erstellt. Sie treten in dieser Form bisher nicht auf und sollen die neue Theorie der Trajektoriensimulationen einführen.

Spricht man von Trajektorien, so muss der entsprechende Phasenraum definiert werden.

Definition 1 (Phasenraum)**.** *Der Phasenraum $\mathbb{P}$ ist die Menge aller Zustände p, die ein Modell der Wirklichkeit annehmen kann.*

Der Phasenraum spiegelt das Abstraktionslevel wider. Stellt man ein System aus lediglich einer Variablen dar, so beinhaltet der Phasenraum alle Belegungen dieser Variablen. Dies können zumindest theoretisch unendlich viele sein. Für die Darstellung einer Ampelkreuzung könnte man eine Variable für jede Spur nehmen, welche die Anzahl der wartenden Autos beinhaltet oder ein komplexes Konstrukt aus Autos, deren Position, Geschwindigkeiten, Wartezeiten und den Ampelphasen.

Definition 2 (Trajektorien)**.** *Sei $a, b \in \mathbb{R}$ und $T : [a, b] \to \mathbb{P}$ eine kontinuierliche zeitliche Anordnung von Zuständen des Phasenraums $\mathbb{P}$. Dann ist T eine Trajektorie und $T_a := T(a)$ der Anfangszustand und $T_b := T(b)$ der Endzustand.*

$\mathbb{T}[a, b]$ ist die Menge aller Trajektorien mit den Grenzen $a, b \in \mathbb{R}$.

Da der Phasenraum in der Regel unendlich ist, beschreibt eine Trajektorie eine Bahnkurve in diesem unendlichen Raum. Praktisch wird eine Trajektorie verwendet, um die Entwicklung eines Modells über die Zeit hinweg darzustellen. Trajektorien können programmiertechnisch nicht exakt erfasst werden, da sie in der Regel unendlich groß sind. Daher werden Trajektorien simuliert.

Definition 3 (Simulation)**.** *Seien $p_0, p_1 \in \mathbb{P}$ zwei Zustände des Phasenraums und $t \in \mathbb{N}$ ein Zeitschritt, so dass p_1 eine mögliche in t Zeitschritten erreichbare Zukunft der dargestellten Wirklichkeit von p_0 ist. Dann ist p_1 ein simulierter Nachfolger von p_0 in t Zeitschritten. Wir schreiben $p_1 = N(p_0, t)$.*

Simulationen bringen eine zeitliche Ordnung in den Phasenraum. Zustände können nun diskrete Zeitabstände zueinander haben. Dabei ist es nicht zwingend, dass alle Zustände zueinander einen zeitlichen Abstand besitzen. Die Wahl der diskreten Zeitabstände unterstützt dabei die Entwicklung mit programmiertechnischen Mitteln, ohne die Genauigkeit der kontinuierlichen Trajektorien zu verlieren.

Definition 4 (Trajektoriensimulation)**.** *Sei $p_0 \in \mathbb{P}, t \in \mathbb{N}$ mit $t > 0$ und sei $T \in \mathbb{T}[0, t]$ eine zufällige Trajektorie mit $T(0) := p_0$ und $T(i + 1) = N(T(i), 1)$. Dann ist*

$$S\{0, \ldots, t\} \to \mathbb{P}, S(x) := T(x)$$

eine Trajektoriensimulation des Zustands p_0 mit t Zeitschritten. $S(p_0, t) \in \mathbb{P}$ beschreibt einen Zustand $p_t := S(t)$ einer Trajektoriensimulation S.

Eine Trajektoriensimulation bietet nun die Möglichkeit, ausgehend von einem Anfangszustand, zufällige in der Zukunft liegende Zustände programmiertechnisch zu bestimmen und über die Wahl der Zeiteinheit den kontinuierlichen Raum beliebig zu approximieren.

Definition 5 (Bewertungsfunktion)**.** *Sei $f : \mathbb{P} \to \mathbb{R}$ eine Funktion, die jedem Element des Phasenraums eine reelle Zahl zuordnet. Dann ist f eine Bewertungsfunktion. Gilt $f(p_0) > f(p_1)$ so ist p_0 unter der Bewertungsfunktion f besser als p_1. Analog gilt p_1 als schlechter unter der Bewertungsfunktion f wenn $f(p_1) < f(p_0)$ und p_1, p_0 als gleich gut unter f sofern $f(p_1) = f(p_0)$.*

Eine Bewertungsfunktion ist also eine Zahl, die einem Zustand zugeordnet wird. Dabei gilt je größer diese Zahl ist, desto besser ist der Zustand. Nachdem Zustände bewertet werden können, kann man die k-besser Relation definieren, um anschließend damit eine Entscheidung zu definieren.

Definition 6 (k-besser Relation)**.** *Seien $x, y \in \mathbb{N}$ zwei Zeitpunkte mit $y > x$ und A, B zwei disjunkte Mengen von Zuständen des Phasenraums und sei $g \in \mathbb{P}$ ein Zustand, der die Gegenwart einer Wirklichkeit beschreibt, dann gilt A ist k-besser als B unter f ($A \overset{k,f}{>} B$), wenn es jeweils k Trajektoriensimulationen $\{S_1^A, ..., S_k^A\}$ und $\{S_1^B, ..., S_k^B\}$ gibt mit $S_i^A(0) = S_i^B(0) = g$, und dass $S_i^A(x) \in A$ und $S_i^B(x) \in B$ für alle $i \in \{1, .., k\}$ gilt:*

$$\sum_{i=1}^{k} \frac{f(S_i^A(y))}{k} > \sum_{i=1}^{k} \frac{f(S_i^B(y))}{k}$$

Gilt $A \overset{\infty,f}{>} B$, so ist die Alternative A besser als die Alternative B unter f.

Definition 7 (Entscheidung)**.** *Die Wahl einer Alternativen, basierend auf der k-besser-Eigenschaft, ist eine Entscheidung.*

Um diese Definition besser zu verstehen, sollte man einen Blick auf Abbildung 3.1 werfen. Die Alternativen sind jetzt zwei beliebige disjunkte Mengen. Praktisch macht es Sinn, diese sich in einer Eigenschaft eines Zustands unterscheiden zu lassen. Man könnte also sagen, A ist die Menge aller Zustände in denen die Eigenschaft $u = 0$ ist und B die Menge aller Zustände, in denen die Eigenschaft $u = 1$ ist.

Wichtig ist, dass die k-besser-Relation nicht irreflexiv ist. Das bedeutet, A kann k-besser als B sein und gleichzeitig kann B k-besser als A sein. $A \overset{\infty}{>} B$ ist jedoch irreflexiv, da der Erwartungswert exakt bestimmt und nicht mehr vom Zufall abhängt. Man kann eine k-bessere Alternative, die jedoch nicht ∞-besser also „besser“ ist als eine falsche Entscheidung bezeichnen. Eine k-bessere Alternative, die zusätzlich auch besser ist, kann man als eine richtige Entscheidung bezeichnen.

Der Zeitpunkt x ist der Anwendungszeitpunkt, also der Zeitpunkt, zu dem die Gegenwart g in A bzw. B über geht. Daher müssen $S_i^A(x) \in A$ und $S_i^B(x) \in B$ liegen. y ist ein weiter in der Zukunft liegender Zeitpunkt, bei dem die Bewertung stattfindet. Obwohl es Sinn machen würde, die komplette Trajektoriensimulation zu bewerten, wird nur der Zustand zum Zeitpunkt y bewertet. Allerdings kann man, sofern die Veränderung einer Eigenschaft wichtig ist, die Entwicklung dieser Eigenschaft in den Zustand aufnehmen und dann in der Bewertungsfunktion verwenden.

Die beiden Summen, die bei der Definition der Entscheidung verglichen werden, spiegeln den Mittelwert der Bewertungsfunktion f zu k Möglichkeiten der Zukunft zum Zeitpunkt y wider. Dabei müssen die Trajektoriensimulationen zum Zeitpunkt x in der Menge A oder B sein, also die Alternativen durchlaufen. Lässt man k gegen ∞ konvergieren, so nähert sich der approximierte Erwartungswert immer mehr dem eigentlichen Erwartungswert an.

Bezieht man dies auf das Beispiel 1 auf Seite 16, so wäre die Alternative Kopf 1-besser als Zahl, da $20 > -25$, wobei 20 und -25 die Ausgänge zweier Trajektoriensimulationen wären. Kopf ist besser als Zahl, da $5 > 2.5$ ist und 5 der Erwartungswert für Kopf und 2.5 der Erwartungswert für Zahl ist. Kopf ist also eine richtige Entscheidung. Allerdings ist die Alternative Zahl auch 1-besser als Kopf, da $30 > 20$ ist. Würde man sich aufgrund der 1-besser Eigenschaft für Zahl entscheiden, wäre dies eine falsche Entscheidung, die dennoch eintreten kann.

3.2 Rahmenwerk für Trajektoriensimulationen

In der Abbildung 3.2 sieht man einen typischen Kreislauf eines intelligenten Systems, welches mit seiner Umgebung interagiert. Zuerst nimmt das System Sensordaten aus einer instrumentalisierten Umgebung auf und verarbeitet diese. Daraus errechnet das System Befehle, die an Aktuatoren gesendet werden, welche die Umgebung entsprechend des Ziels der Künstlichen Intelligenz verändern.

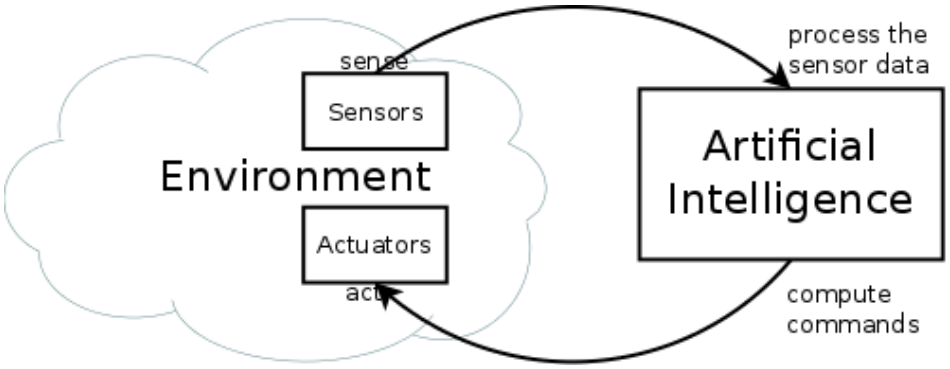


Abbildung 3.2: Konzept einer typischen Künstlichen Intelligenz

Das Konzept der hier beschriebenen künstlichen Intelligenz gliedert sich in zwei unabhängige Prozesse. Zum einen die Synchronisierung und zum anderen die Entscheidungen.

Bei der Synchronisierung wird ein digitales Modell der über die Sensoren wahrnehmbaren Wirklichkeit erzeugt. Dieses Modell muss stetig angepasst und aktualisiert werden. Dazu erwartet das System Sensorevents, welche es verarbeitet und so sein digitales Modell so aktuell wie möglich hält. Zusätzlich zu den Events können Erfahrungen und Wissensdatenbanken hinzugezogen werden.

Sobald ein synchrones digitales Modell vorhanden ist, kann das System zwischen mehreren gegebenen Alternativen auswählen, also eine Entscheidung treffen. Die Alternativen werden beschrieben durch Änderungen am Modell, die zu einer spezifizierten Zeit x in der Zukunft durchgeführt werden sollen. Um die Alternativen zu bewerten, erzeugt der Entscheidungsprozess für jede Alternative die gleiche Anzahl an Kopien des synchronen digitalen Modells und übergibt diese an den Zukunftssimulator. Dieser berechnet mit Hilfe von Zufallsgeneratoren die Zukunft des übergebenen Modells und führt zur spezifizierten Zeit eine Änderung durch. Dadurch liegen dem System mehrere mögliche zukünftige Modelle für jede Alternative vor. Diese werden an einen Bewertungsprozess übergeben. Dieser Bewertungsprozess errechnet die durchschnittliche Punktzahl der verschiedenen Alternativen und gibt die mit der höchsten durchschnittlichen Punktzahl an den Ausführungsprozess weiter. Dieser führt die selben Änderungen in der realen Welt durch, indem er dem Aktuator ein entsprechendes Event übermittelt. Veranschaulicht wird dieses Konzept in der Abbildung 3.3.

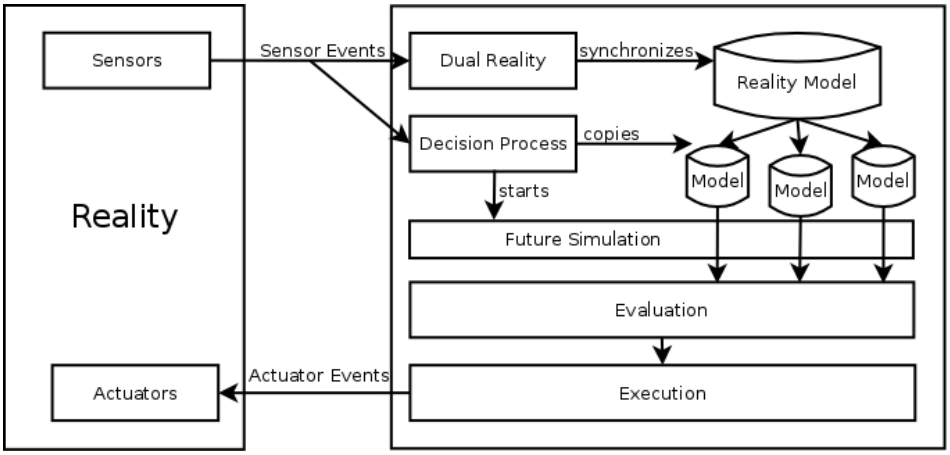


Abbildung 3.3: Konzept des Trajektorien Simulation Decision Frameworks (TSDF)

Benötigte Komponenten für eine Entscheidung

Die folgenden Komponenten müssen von einem Programmierer umgesetzt werden, um intelligente Entscheidungen mit Hilfe des Rahmenwerks fällen zu können:

- **Ein Modell**
  Für die Situation muss ein abstraktes Modell herangezogen werden. Wie detailliert dieses sein soll ist abhängig vom Prozess, der entschieden werden muss. Das Model kann von einer einfachen Variable bis zu einem komplexen objektorientierten Gebilde reichen.

- **Eine Synchronisationseinheit**
  Empfängt Sensordaten und verändert entsprechend dieser das Model. Zusätzlich kann sie Informationen aus Datenbanken oder anderen Quellen heranziehen um ein besseres Model zu erzeugen.

- **Eine Simulationseinheit**
  Eine Simulationseinheit nimmt eine Kopie des Modells, zusammen mit einer Zeit und berechnet eine zufällige aber realistische Zukunft des Models entsprechend der Zeit.

- **Eine Bewertungsfunktion**
  Eine Bewertungsfunktion nimmt ein Modell und liefert eine Zahl zurück. Je größer diese Zahl ist, desto besser ist das Modell. Die Bewertungsfunktion kann alle im Modell vorhandenen Werte benutzen, um eine geeignete Punktzahl zu errechnen. Es kann vorteilhaft sein, spezielle Werte extra in das Modell und den Simulator aufzunehmen, um die Laufzeit zu verringern.

- **Die Alternativen**
  Eine Alternative ist eine Veränderung am Modell zu einer gewissen Zeit. Auch wenn man dem System beliebige Änderungen zur Auswahl stellen kann, macht es nur Sinn Alternativen, die durch Aktuatoren umgesetzt werden können, zur Entscheidung zu stellen. Das Rahmenwerk kann nur zwischen spezifizierten Alternativen entscheiden.

- **Eine Ausführungseinheit**
  Die Ausführungseinheit setzt die beste errechnete Alternative in der Realität um.

In der praktischen Umsetzung, können Modell, Synchronisationseinheit und Simulator zu einer Einheit verschmelzen. Der Simulator arbeitet dann auf einem internen Modell und bekommt die Synchronisationsdaten und baut diese ein, um eine aktuelle Gegenwart halten zu können. Da die Alternativen eine Änderung am Modell beinhalten, können sie ebenfalls das entsprechende Aktuatorenevent beinhalten. Dadurch würde eine separate Ausführungseinheit wegfallen.

Aktuatoren haben je nach Situation eine mehr oder minder starke Auswirkung auf die Umgebung. Besonders, wenn diese Aktuatoren mit Menschen interagieren, können jene die Aktuatoren ignorieren. Im Kassenszenario werden Monitore aufgestellt, die eine Kasse vorschlagen. Es ist zu erwarten, dass viele Kunden diese Monitore ignorieren und selbst entscheiden, wo sie sich anstellen. Im Gegensatz dazu werden Ampeln von Autofahrern nicht ignoriert. Soll die KI einen positiven Einfluss auf die Umgebung ausüben, so kann sie diesen leichter erreichen, wenn der Einfluss der Aktuatoren groß ist. Dies ist der Hauptgrund, warum zum Testen des Systems die Ampelsteuerung und nicht das Einkaufsszenario verwendet wurde.

Zur Synchronisierung muss die Wirklichkeit abstrahiert werden. Zusätzlich können bei der Verwendung von Sensoren Fehler auftreten. Die Implementierung des Simulators kann ebenfalls nicht alle Details der Wirklichkeit abbilden. Dadurch ist es möglich, dass der approximierte Erwartungswert ebenfalls fehlerbehaftet ist und somit die Entscheidungen an Qualität verlieren. Es muss daher darauf geachtet werden, dass die Abweichung zur Wirklichkeit in einem verträglichen Rahmen geschieht. Allerdings kann keine Aussage darüber getroffen werden, wie groß diese Abweichung sein darf und kann nur durch Tests entschieden werden.

3.3 Trajektoriensimulation zur Ampelsteuerung

Das Rahmenwerk der Trajektoriensimulation soll nun zur Steuerung einer Ampelkreuzung konzipiert werden. Für den daraus resultierenden „electronical Traffic Arbitration Service“ (kurz eTAS) können die Sensoren herangezogen werden: Magnetresonanzschleifen, Flusskameras, Knöpfe für Fußgänger und Lichtschranken. Theoretisch sind alle Sensoren denkbar, die Informationen über die Position, Richtung und Geschwindigkeit der Autos und Fußgänger an der Ampel erfassen können. Als Aktuator wird die Phasenkontrolleinheit verwendet, die zwischen verschiedenen Ampelphasen umstellt.

Um Entscheidungen treffen zu können, braucht man die beschriebenen Komponenten. Für eine Verkehrssteuerung könnten diese wie folgt spezifiziert werden:

- Als Modell verwendet diese Arbeit ein objektorientiertes Klassenkonstrukt, welches Autos, Spuren, Magnetresonanzschleifen, Ampeln und Ampelphasen mit all ihren üblichen Parametern heranzieht. Es wird also ein mikroskopisches Modell zur Simulation verwendet. Dieses Modell kann von der Wirklichkeit abweichen.

- Zur Synchronisation wurde jede unabhängige Spur, sobald sie entsteht, mit einer eigenen Magnetresonanzschleife ausgestattet, so dass exakt erfasst werden kann, wenn ein Auto auf diese Spur fährt. Autos die später auf diese Spur wechseln, können nicht mehr erfasst werden und bleiben im Modell auf der ursprünglichen Spur. Wird die Resonanzschleife aktiviert, so wird im virtuellen Modell ein Auto auf dieser Spur eingefügt, sofern dies ohne Zusammenstoß möglich ist. Die Autos benutzen immer ihre maximale Beschleunigung, solange sie dabei in den nächsten Sekunden nicht mit einem anderen Auto zusammen stoßen oder die zugelassene Höchstgeschwindigkeit überschreiten. Gäbe es einen Zusammenstoß, so verringert das Auto auf der Spur mit dem geringeren Vorfahrtrecht seine Beschleunigung schrittweise bis zur maximalen Bremswirkung. Kann ein Zusammenstoß dennoch nicht verhindert werden, so wird die Geschwindigkeit des Autos auf 0 gesetzt. Blockiert ein Auto mit der Geschwindigkeit 0 ein Auto mit Vorfahrt, so setzt dieses ebenfalls seine Beschleunigung schrittweise bis auf die maximale Bremswirkung herab. Kann eine Kollision nicht verhindert werden, wird die Geschwindigkeit auf 0 gesetzt. Um eine Blockierung des Modells zu verhindern, werden Autos, die länger als 5 Minuten die Geschwindigkeit 0 haben, aus dem Modell entfernt. Zusätzlich erfasst die Synchronisation, wie viele Autos in den letzten zehn Minuten von einer Resonanzschleife erfasst wurden und benutzt dies als Referenz während der Simulation. Ebenfalls wird die aktuelle Ampelphase erfasst.

- Der Simulator berechnet das gleiche Beschleunigungsverhalten, wie die Synchronisation. Erreicht ein Auto eine Spurtrennung, so wird die Entscheidung mit Hilfe einer statistisch ermittelten Wahrscheinlichkeit gefällt. Da die Erfassung der Autos erst nach dem Spurenwechsel erfolgt, passt sich diese statistische Wahrscheinlichkeit automatisch der Realität an. Da die Autos auf der Kreuzung selbst jedoch auch eine Richtungsentscheidung treffen müssen, kann hier bei den Simulationen eine Differenz zur Wirklichkeit entstehen, da über Sensoren nicht erfasst wird, in welche Richtung ein Auto tatsächlich fährt. Da das Auto zu diesem Zeitpunkt die Ampel jedoch passiert hat, hat dies noch einen blockierenden Einfluss auf

die Gegenspur. Mit Verkehrsüberwachungskameras könnte dies jedoch weiter optimiert werden. Neue Autos werden in einer Simulation, entsprechend der von den Resonanzschleifen erfassten Autorate, eingefügt.

- Zur Bewertung werden die Geschwindigkeiten der Fahrzeuge positiv, so wie die Wartezeiten der Autos quadratisch negativ verwendet und mit geeigneten Faktoren versehen.

- Als Alternativen werden alle zur Verfügung stehenden Ampelphasen umgesetzt. Die Änderung einer Ampelphase bedeutet, alle Ampeln auf rot zu stellen und anschließend alle zu dieser Phase gehörenden Ampeln auf grün zu stellen. Natürlich kann eine Ampel zu mehreren Phasen gehören. Die Umschaltung von Grün auf Rot und umgekehrt wird mit einer Gelb-Phase und einer Rot-Gelb-Phase mit entsprechenden Umschaltzeiten verlängert.

- Die Ausführungseinheit gibt die errechnete Ampelphase an die Steuerungseinheit der echten Ampel weiter und schaltet auch die Ampelphase in der Virtuellen Realität entsprechend um.

Die Entscheidung der Ampelphase wird in einem regelmäßigen Intervall durchgeführt. Dieses Intervall sollte länger sein, als der Computer für eine Zukunftssimulation benötigt, damit das System nicht überlastet wird. Es sollte aber auch nicht zu lang sein, da sonst die Ampel zu selten umgeschaltet wird. Die Befürchtung, dass das System eine Ampel zu oft schaltet und somit die Kreuzung blockiert, ist unbegründet, sofern der Simulator Anfahrtszeiten und die Zeit für die Phasenumschaltung mit simuliert und die Wartezeit der Autos evaluiert wird. Da das System die Wartezeit minimiert, wird es sich oft auch die für die aktive Phase entscheiden und daher nicht umschalten, sofern dies nicht benötigt wird.

Mögliche Evaluationsparameter in der Übersicht:

- Wartezeit der Autos (negativ)
  Verwendet man die Wartezeit für Autos, führt dies dazu, dass das System diese minimiert. Lässt man die Wartezeit quadratisch in die Evaluation eingehen, so kann man verhindern, dass Autos wegen anderer Parameter ignoriert werden und deshalb sehr lange stehen würden.

- Geschwindigkeit der Autos (positiv)
  Nimmt man die Geschwindigkeit hinzu, so werden Autos, die bereits mit hoher Geschwindigkeit auf die Kreuzung zufahren, durchgelassen. Dies sollte jedoch nur mit quadratischer Wartezeit kombiniert werden, da sonst der Fall eintreten kann, dass vereinzelt wartende Autos lange ignoriert werden.

- Anzahl der Bremsvorgänge und Beschleunigungsvorgänge (negativ)
  Dies reduziert die Frustration der Autofahrer und wäre vorteilhaft für die Umwelt, da das System dann versuchen würde diese zu minimieren.

- Dauer der Rotphase (negativ)
  Lässt man die Zeit einer Rotphase negativ auf die Bewertung einfließen, so verhindert man, dass eine Ampel dauerhaft grün bleibt und mögliche, durch die Sensoren

nicht erkannte Fahrzeuge, nicht fahren können. Würden alle Sensoren ausfallen, so würde sich das System mit dieser Bewertung dann wie eine zeitorientierte Ampelphasensteuerung verhalten.

- Geschwindigkeit spezieller Fahrzeuge (positiv)
  Hat man Sensoren, welche spezielle Fahrzeuge wie Busse oder Einsatzfahrzeuge erkennen können, so kann man deren Geschwindigkeit positiv einfließen lassen und somit das System anweisen, diese zu bevorzugen.

3.4 Kundenleitsystem

Um das Konzept der Kassenwahlunterstützung eCASE besser verstehen zu können, müssen mögliche Sensoren genauer betrachtet werden. Allgemein haben Sensoren unterschiedliche Genauigkeiten, Wirkungsbereiche und eine unterschiedliche Kundenabschreckung. Zusätzlich spielen auch die Kosten eine entscheidende Rolle. Im folgenden werden also kurz die verschiedenen denkbaren Sensoren genauer betrachtet:

- **Kassiererschätzung**
  Eine Kassierereinschätzung ist kein Sensor im eigentlichen Sinne. Die Kassierer sollen dabei im Laufe einer Kundenabwicklung schätzen, wie viele Kunden an der Kasse anstehen. Vor allem bei kleinen Zahlen ist diese Schätzung sehr genau und sehr kostengünstig umsetzbar. Eine Kundenabschreckung ist nicht vorhanden, da sie den Vorgang nicht bemerken und selbst wenn sie davon wissen, spielt es für ihre Privatsphäre keine Rolle, da sie lediglich als Zahl erfasst werden. Dies ist in der Bundesrepublik Deutschland auch mit dem aktuellen Datenschutzgesetzen konform. Die Daten, wie viele Menschen an der Kasse anstehen, sind für eine Kassenwahlunterstützung wichtig, denn sie sind ein entscheidender Faktor in der Wartezeiteinschätzung.

- **Kameras**
  Kameras können erkennen, wo sich Menschen im Laden aufhalten. Zusammen mit spezieller Software können sie die Menschen mit einer Kennung versehen und die Position, Blickrichtung und Bewegungen exakt [34, 32, 35, 33] verfolgt werden[3]. Da die Kunden nicht wissen, was die Software macht, können sie Kameras, die zur Fluss- und Positionsanalyse verwendet werden, nicht von normalen Überwachungskameras unterscheiden. Daher können sie beim Kunden ein Überwachungsgefühl verursachen. Wang et al. beschreiben ferner, dass Kameras verwendet werden können, um menschliche Intentionen zu ermitteln und zu bestimmen [56, 11]. Außerdem ist es möglich, die Bewegung von Menschen vorher zu sagen, was für die Simulation in Kaufhäusern ein entscheidender Vorteil ist [55]. Die Erkennung von Menschen ist auch in unübersichtlichen Menschenmengen möglich [48]. Kameras sind also insgesamt sehr gute Sensoren und haben vielseitige Möglichkeiten. Der Nachteil ist, dass Kameras mit ausreichender Geschwindigkeit und Qualität in großer Zahl angebracht und dazu große Datenmengen ausgewertet werden müssen, was sich erheblich negativ auf den Preis auswirkt.

- **Lichtschranken**
  Lichtschranken können verwendet werden, um zu zählen, wie viele Menschen einen Laden betreten oder verlassen. Sie sind allerdings nicht sehr genau, da sie je nach angebrachter Position, mehrere Menschen nicht unterscheiden können oder alternativ die Beine und Einkaufswagen mitzählen. Lichtschranken sind aber sehr günstig und können daher für eine grobe Einschätzung gut verwendet werden. Da Lichtschranken keine persönlichen Daten erfassen, werden sie von Kunden nicht als unangenehm empfunden.

[3] `https://www.youtube.com/watch?v=PLPbfOa1DlE`, Video abgerufen am 12. Mai 2014, zeigt die Genauigkeit und Möglichkeiten von Kameras

- **RFID-Antennen**
  RFID-Antennen können an mehreren Stellen verwendet werden. Zum einen zur Positionierung und zum anderen zur Produkterkennung. Problematisch bei der Verwendung von RFID sind Gegenstände aus Metall oder mit flüssigem Inhalt, da diese die elektromagnetische Induktion verhindern. Solche problematischen Gegenstände können nur schwierig mit RFID-Tags versehen werden. Dennoch wären für eine generelle Produktabschätzung RFID-Tags eine gute Erweiterung, um zusätzliche Daten für das intelligente System bereitzustellen. Nachteilig ist vor allem der Preis. Der Stückpreis beträgt etwa 19 Eurocent[4], für eine allgemeine Ausstattung der Lebensmittel ist dies zu teuer. Auch bei der Entsorgung der Verpackung würden weitere Probleme entstehen, da die RFID-Tags elektronische Bauteile beinhalten. Zur ungefähren Ortung von Einkaufswagen wäre eine Verwendung denkbar. Die Abschreckung ist eher gering, da die Tags nahezu unsichtbar sind und keine persönlichen Informationen über den Kunden preisgeben, sofern sie an der Kasse beim Bezahlen zerstört werden. Zur Zerstörung besitzen die RFID-Tags ein spezielles Signal, welches über Funk bei einem Bezahlvorgang übertragen werden könnte. Würden die RFID-Tags nicht zerstört werden, so könnte man die Produktverpackungen den Kunden zu späteren Zeitpunkten zuordnen.

- **Bluetooth**
  Mit der Einführung des Bluetooth Low Energy Protokolls[5], können die Positionen von Einkaufswagen oder Kunden auf den Meter genau bestimmt werden, sofern diese einen BLE-Token mit sich führen. Die Instrumentalisierung einer Umgebung mit speziellen Bluetooth Antennen ist jedoch sehr kostenintensiv. Die Laufzeitkosten sind jedoch vernachlässigbar und reduzieren sich auf jährliche Batteriewechsel bei den Tokens, sowie den geringen Stromverbrauch der Antennen. Eine Abschreckung bei der Ortung der Einkaufswagen im Laden findet kaum statt, da keine persönlichen Daten erfasst werden. Würde man jedoch die Mobiltelefone der Kunden orten, könnte dies eine geringe Abschreckung zur Folge haben.

Technik	Genauigkeit	Abschreckung	Kosten	Nutzen
Kassiererschätzung	mittel	gering	gering	hoch
Kameras	hoch	hoch	mittel	mittel
Lichtschranken	gering	gering	gering	gering
RFID-Position	mittel	mittel	mittel	gering
RFID-Produkte	mittel	gering	hoch	gering
Bluetooth	hoch	mittel	hoch	mittel

Tabelle 3.1: Übersicht Genauigkeit, Abschreckung, Kosten und Nutzen

Tabelle 3.1 zeigt eine Schätzung der Genauigkeit, Abschreckung, Kosten und Nutzen, wenn man diese als Datenquellen für die Kassenempfehlungen nutzen möchte. Die Ein-

[4] `http://www.pressrelations.de/new/standard/dereferrer.cfm?r=217758` Mo. 26. Mai 2014
[5] `http://de.wikipedia.org/wiki/Bluetooth_Low_Energy` Informationen über Bluetooth Low Energy kurz BLE

schätzungen für Genauigkeit und Nutzen ergeben sich durch Simulationen, die im Rahmen der Entwicklung durchgeführt wurden. Die Informationen über die Abschreckung sind eine persönliche Einschätzung und die Kosten ergeben sich durch Hochrechnung der Anschaffungs- und Betriebskosten.

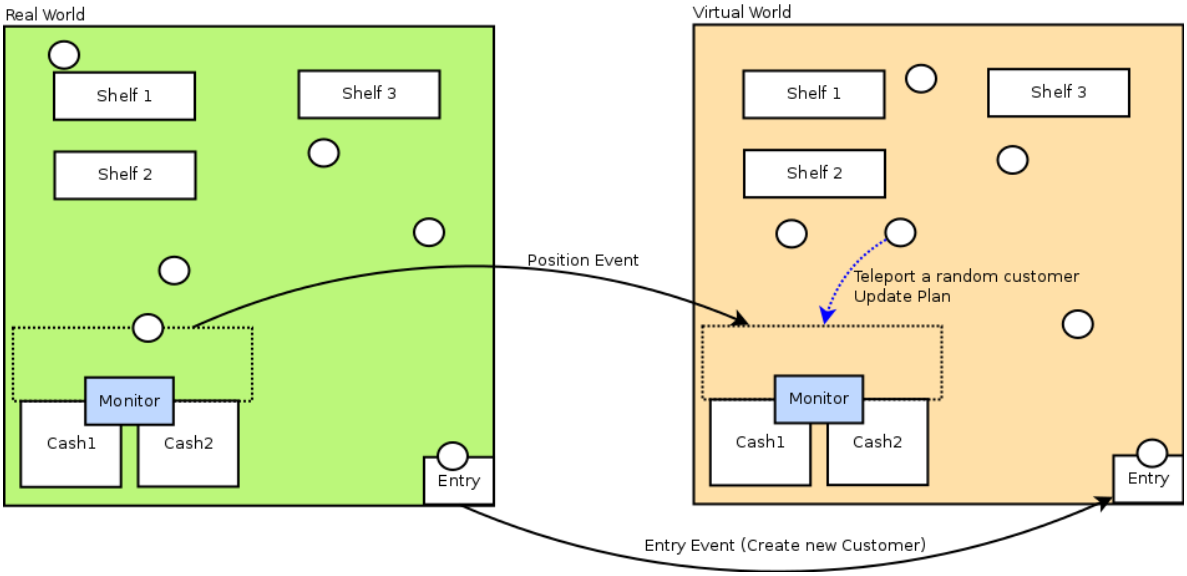


Abbildung 3.4: Konzept der Synchronisierung von Positionssensordaten

Die Abbildung 3.4 zeigt, wie man mit ungenauen Positionsdaten dennoch eine gute Synchronisierung erzielen kann. Wird ein neuer Kunde im Eingangsbereich erkannt, so wird ebenfalls ein Kunde in der virtuellen Welt erzeugt. Da keine Informationen über die Einkaufsliste und Pläne vorliegen, verhält sich dieser Kunde entsprechend einer zufällig generierten Einkaufsliste. Diese Einkaufslisten sollten mit statistischen Werten des Ladens untermauert werden. Erhält das System ein Positionsevent, so wird ein zufälliger Kunde im virtuellen System an diese Stelle verschoben und sein Plan angepasst, sofern neue Informationen darüber vorliegen.

Befinden sich viele Kunden im Laden und man verfügt über Kassiererschätzungen, so können Kassen in der virtuellen Welt, die nicht der Kassiererschätzung entsprechen bevorzugt oder gesperrt werden, so dass sich die Zahl der anstehenden Kunden automatisch anpasst. Dies wird im Rahmen der Arbeit verwendet und liefert eine Fehlerrate von weniger als 10% an den Kassen[6].

Zur Kassenempfehlung eignen sich zwei Ansätze. Beim ersten Ansatz werden Monitore an den Hauptwegen aufgestellt, welche den Kunden Kassen mit geringer Wartezeit empfehlen. Dafür werden im Simulator ebenfalls Monitore platziert, die einen gewissen Einflussbereich haben. Betritt ein virtueller Kunde auf dem Weg zur Kasse den Einflussbereich eines Monitors, so wird er diese Empfehlung mit einer spezifizierten Wahrscheinlichkeit übernehmen. Die benötigte Wahrscheinlichkeit, damit eine Optimierung stattfindet, ist ein wichtiger Parameter und soll ermittelt werden. Ferner soll ermittelt werden, wo geeignete Positionen für die Monitore sind, in dem verschiedene Szenarien getestet werden. Der zweite Ansatz errechnet für einen Kunden individuell eine geeignete Kasse, nachdem er mit seinem Smartphone eine Anfrage an das System gestellt hat.

[6]Dies beinhaltet die Fehleinschätzung der Kassierer sowie die Abweichung zur realen Situation.

Hier übernimmt der Kunde immer die Empfehlung des Mobiltelefons. Es muss jedoch festgestellt werden, welcher Mengenanteil von Kunden das System benutzen muss, um eine Optimierung zu erreichen. Ferner wird getestet, ob eine bessere Optimierung erreicht wird, wenn der Kunde zusätzliche Informationen über Warenmenge (Viel, Mittel, Wenig) und Zahlungsmittel (Bar, Kreditkarte) angibt.

Zur Ermittelung der optimalen Kassen werden für beide Varianten ebenfalls verschiedene Methoden zugrunde gelegt. Zur Berechnung, welche Kasse auf den Monitoren beworben wird, gibt es $(\#Monitor \cdot \#Kasse)$ Alternativen. Je nach Anzahl der Kassen und Monitore sind dies zu viele, um mehrere Simulationen für jede Alternative durchzuführen. Daher werden nur bestimmte Kombinationen als Alternativen bereit gestellt:

$1\times$ Die Aktuelle Konfiguration

$x\times$ Die Aktuelle Konfiguration mit Veränderungen

$y\times$ Zufällige Konfigurationen

Das System muss also jedes Mal nur $k \cdot (x + y + 1)$ Alternativen simulieren. Die Werte für x und y müssen an die Geschwindigkeit des Computers angepasst werden und die Anzahl der Veränderungen sollte nicht mehr als $1\% - 10\%$ der offenen Kassen betragen. Die so spezifizierte Entscheidung muss das System regelmäßig (alle 10-60 Sekunden) fällen und die Monitore entsprechend anpassen. Insgesamt ergibt sich dadurch ein System, dass einem genetischen Algorithmus gleicht. Wird eine bessere Konfiguration für die aktuelle Situation gefunden, so wird diese übernommen, wenn nicht bleibt die alte aktiv, da sie immer als erste Alternative zur Verfügung steht.

Für die Berechnung mit mobilen Endgeräten reagiert das System auf die Anfrage eines Kunden, simuliert alle offenen Kassen als Alternative und liefert dem Kunden die Kasse mit dem besten Ergebnis zurück. Der Vorteil dieses Systems entsteht, wenn mehrere Kunden nahezu gleichzeitig anfragen. Der erste Kunde bekommt die beste Wahl, für den nächsten der anfragt, wird aber bereits der andere Kunde auf dem Weg zur empfohlenen Kasse mit eingerechnet. Daher ist das Ergebnis exakter, als immer nur die Kasse mit der kürzesten Schlange zu empfehlen.

Mögliche Evaluationsparameter in der Übersicht:

- Wartezeit der Kunden an den Kassen (negativ)
  Die Wartezeit der Kunden kann negativ aufsummiert werden. Das System versucht dann, die Wartezeit zu reduzieren. Dabei wird die Zeit vom Anstellen bis zum Abschluss des Bezahlvorgangs erfasst.

- Wegzeit der Kunden bis zur Kasse (negativ)
  Wird dem Kunden eine weit entfernte Kasse vorgeschlagen, so benötigt er eine gewisse Wegzeit. Diese Zeit sollte ebenfalls negativ in die Evaluation einfließen, damit möglichst naheliegende Kassen empfohlen werden.

3.5 Filialleiterunterstützung bei der Kassenöffnung

Die Verwendung des Systems zur Unterstützung der Entscheidung, ob neue Kassen geöffnet werden oder nicht, ist ein optionales Ziel dieser Masterarbeit. Die Schwierigkeit bei dieser Erweiterung ist, die Wartezeit geeignet gegen die Kosten einer Kassenöffnung abzuschätzen. Das Problem dabei ist, dass durch eine geöffnete Kasse reale Kosten entstehen, da ein Mitarbeiter diese Tätigkeit übernehmen muss. Die Wartezeit der Kunden dagegen kann nicht unmittelbar in einen Umsatzrückgang umgerechnet werden. Dieser entsteht erst indirekt, da ein Kunde nach einer langen Wartezeit sich entschließen könnte, diesen Laden nicht mehr zu besuchen. Daher ist die Wartezeit der entscheidende Schlüssel. Die Entscheidung eine Kasse zu öffnen, ist ferner mit einer relativ hohen Verzögerung versehen, da die Kassierer informiert werden und sich auf den Weg zur Kasse begeben müssen. Kassen können also nicht ständig geöffnet und wieder geschlossen werden.

Zur Entscheidung müssen Alternativen bestimmt werden. Die Alternativen entstehen durch die einzelne Invertierung des Zustands aller Kassen zuzüglich der aktuellen Situation. Die Anzahl der Alternativen ist also um genau eins größer als die Anzahl der Kassen. Ist zum Beispiel von drei Kassen die erste Kasse geöffnet (100), werden folgende Alternativen getestet: 100 - der Aktuelle Zustand, 000 - die Invertierung der ersten Kasse, 110 - die Invertierung der zweiten Kasse und 101 - die Invertierung der letzten Kasse. Konkret bedeutet dies, dass maximal eine offene Kasse geschlossen wird oder maximal eine geschlossene Kasse geöffnet wird oder der aktuelle Zustand beibehalten wird. Zusätzlich wird bewertet, welche Kasse geöffnet oder geschlossen werden soll.

Eine geeignete Simulationszeit muss höher sein als die Zeit, die benötigt wird, um eine Kasse zu öffnen und außerdem genügend Zeit beinhalten, dass sich die Situation etabliert, um die Bewertung durchzuführen. Die Anwendungszeit x wird daher auf 5 Minuten festgesetzt und die Simulationszeit y insgesamt auf 15 Minuten. Die Entscheidung ist also relativ teuer im Vergleich zu den Entscheidungen bei eTAS und eCASE.

Als geeigneter Evaluationsparameter bietet sich die quadratische Abweichung von einer spezifizierten erwarteten durchschnittlichen Wartezeit an. Möchte man also zum Beispiel, dass die Kunden optimaler Weise etwa 3 Minuten warten, so beträgt die Punktzahl in der Evaluation:

$$-\left|180000 - \frac{1}{|K|}\sum_{k\in K} Wartezeit(k)\right|^2$$

Dabei ist die Menge K die Menge aller wartenden Kunden. Gibt es keine wartenden Kunden, so wird einfach die negative Anzahl an offenen Kassen zurückgegeben, damit diese schrittweise geschlossen werden. Sind keine Kassen geöffnet, so wird $-\infty$ zurückgegeben, um zu verhindern, dass alle Kassen geschlossen werden. Da $-\infty$ keine erlaubte Rückgabe für das Entscheidungsrahmenwerk ist, muss eine geeignet große negative Zahl zurückgegeben werden.

3.6 Verwendung Simulierter Realitäten

In der Einleitung wurde bereits angesprochen, dass diese Arbeit nicht mit echten instrumentalisierten Umgebungen arbeitet, da die spezielle Instrumentalisierung mit Sensoren und Aktuatoren für diese Arbeit zu teuer ist und derartige Umgebungen nicht in benötigter Form verfügbar sind. Zur Lösung dieses Problems wird diese Arbeit die echten Umgebungen ebenfalls simulieren und diese mit virtuellen Sensoren und Aktuatoren ausstatten. Diese Simulierte Realität wird gestaltet wie eine echte Umgebung und die Sensoren und Aktuatoren werden ausschließlich über das Netzwerk angesprochen. Dies ermöglicht es, die Arbeit schnell auf eine echte Umgebung umzustellen, in dem man lediglich eine Sensoren- und Aktuatorenschnittstelle implementiert.

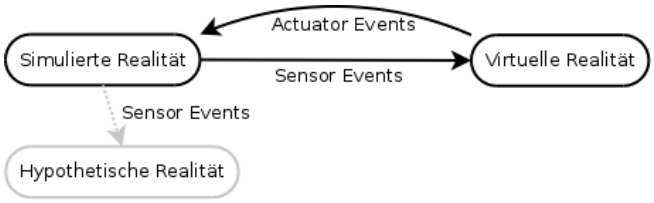


Abbildung 3.5: Darstellung des Datenfluss zwischen Simulierter Realität, Virtueller Realität und Hypothetischer Realität

Zur Differenzierung (vgl. Abbildung 3.5) werden im folgenden die Begriffe **Simulierte Realität**, für die Simulation verwendet, die eigentlich eine echte physische Umgebung seien soll und **Virtuelle Realität**, für die durch Sensordaten synchronisierte Modelldarstellung der Simulierten Realität. Zusätzlich wird für die Ampelsteuerung noch eine **Hypothetische Realität** erzeugt, die mit den gleichen Sensordaten synchronisiert wird, jedoch eine Zeitphasenschaltung besitzt, um einen direkten Vergleich zur Simulierten Realität zu haben.

Für die Ampelsteuerung wird als Vorbild eine Abstraktion der Kreuzung Kaiserstraße - Kolbenholz in Schafbrücke 66121 Saarbrücken[7] gewählt. Diese Kreuzung ist aus mehreren Gründen gut geeignet:

1. Die Kreuzung verfügt über jeweils eine eigene Spur für Linksabbieger aus Westen, Süden und Osten. Dadurch können Spurwechsel und Staus besser produziert werden. Besonders kritisch sind hier die Straßen aus Süden und Osten, da hier die Linksabbiegerspuren, wenn sie voll sind, die Spuren der geradeaus Fahrenden blockieren können. Kommen also viele Linksabbieger und die Ampelphasen für diese sind zu kurz, so blockiert die komplette Kreuzung. Zusätzlich haben in der aktuellen Zeitschaltung die Linksabbieger aus Osten nur eine sehr kurze isolierte Grünphase, sodass bei viel Verkehr nicht mehr als 4 bis 5 Autos links abbiegen können.

2. Die Bahnstraße, die aus Norden in die Kreuzung mündet, ist sehr wenig befahren. Sie besitzt zwar eine Kontaktschleife, diese scheint aber keinen Einfluss auf die Phasenschaltung zu nehmen, so dass diese immer wieder auf Grün schaltet, ohne dass dort überhaupt ein Auto wartet. Dies ist verlorene Zeit für den Hauptverkehr, der von Westen nach Osten und umgekehrt fließt. Eine intelligente Schaltung

[7] GPS Koordinaten: 49.225141, 7.044784

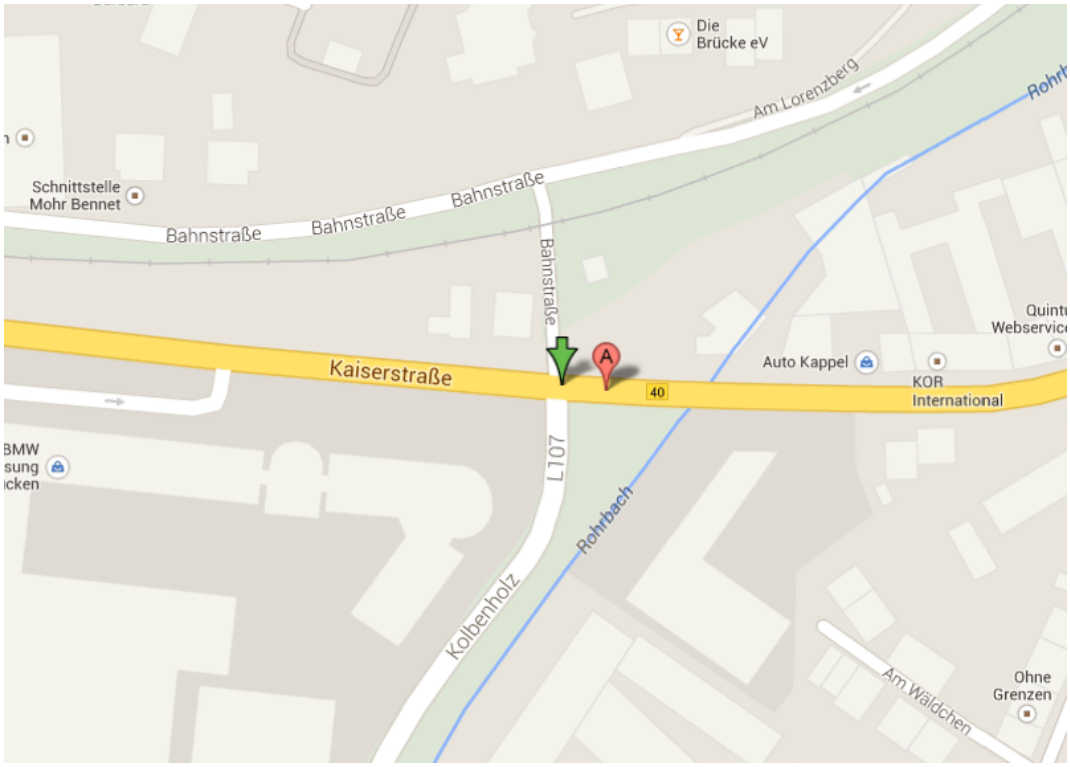


Abbildung 3.6: Darstellung der für die Ampelsteuerung gewählten Kreuzung auf Google Maps

könnte diese Ampel, nur wenn sie benötigt wird, auf Grün schalten und so eine Optimierung erzielen.

3. Die Spuren aus Westen sind sehr lang (mehrere hundert Meter) und können sich praktisch nicht gegenseitig blockieren. Dies erhöht die Vielseitigkeit dieser Kreuzung.

4. Die Kreuzung besitzt zwar Ampeln für Fußgänger, die Anzahl dieser ist jedoch vernachlässigbar gering und da Fußgänger im vorgestellten System nicht berücksichtigt werden, kann so die Optimierung besser gemessen werden.

Um die Messung durchführen zu können, wird zusätzlich zur Simulierten Realität und der Virtuellen Realität eine Hypothetische Realität synchronisiert, in der die Ampel mit einer Zeitschaltung versehen ist. Diese Hypothetische Realität reagiert ebenfalls auf die Sensordaten und erzeugt daher die gleichen Autos, wie die Virtuelle Realität und entsprechend der Simulierten Realität. Alle drei Realitäten sollen graphisch dargestellt werden, so dass man die Verkehrsentwicklung in Echtzeit vergleichen kann.

Zur Umsetzung des Kassenleitsystems eCASE wird als Vorbild die untere Etage des Globus Warenhauses in Güdingen Saarbrücken verwendet (vgl. Abbildung 3.7). Ursprünglich sollten aus alten Kasseninformationen Einkaufsstatistiken generiert werden, um realistische Laufwege der Kunden realisieren zu können. Allerdings konnten die bei Globus angefragten Informationen nicht übermittelt werden und deshalb musste das System angepasst werden. Da nun keine genaue Verteilung der Einkäufe bekannt ist, macht es auch keinen Sinn die existierenden Waren genau zu positionieren, obwohl

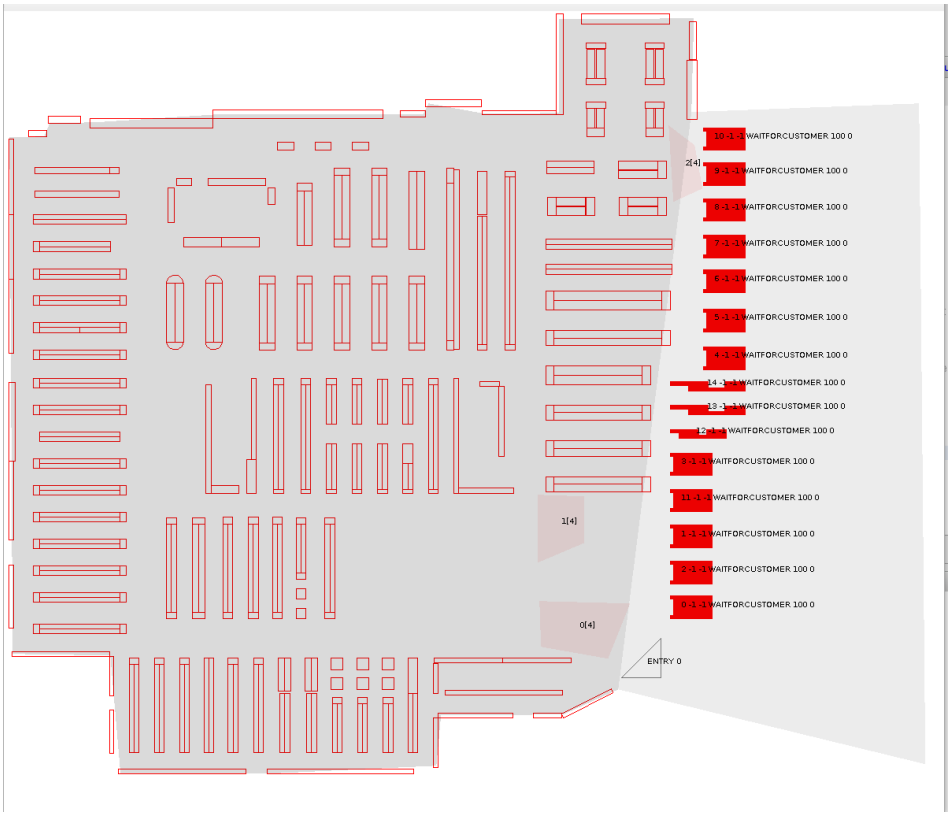


Abbildung 3.7: Darstellung der unteren Etage vom Globus Warenhaus Güdingen

diese Daten verfügbar wären. Daher sollen zufällige Produkte im Laden an geeigneten Stellen platziert werden und die Simulierte Realität erzeugt für die Kunden daraus zufällige Einkaufslisten, die dann abgearbeitet werden. Dadurch ergeben sich unrealistische Laufwege und die Richtung des Andrangs auf die Kassen verändert sich gegenüber dem realen Markt in Güdingen.

Simulierte Realitäten müssen zeitlich mit den anderen Realitäten synchronisiert werden. Normal dient als Referenz der Systeme die Uhrzeit, da jedoch die Simulierte Realität eine eigene Uhr als Referenz hat, muss die Uhrzeit regelmäßig mit den anderen Realitäten abgeglichen werden. Dies hat außerdem den Vorteil, dass man für Tests die Realität im Zeitraffer ablaufen lassen kann und sich die beiden anderen Realitäten automatisch daran orientieren.

4 Vergleich mit anderen intelligenten Systemen

Dieses Kapitel vergleicht die vorgestellte Trajektoriensimulation mit anderen bekannten KI Systemen. Dabei wird nicht versucht die direkte Qualität zu vergleichen, da ein direkter Vergleich oft nicht möglich oder sehr zeitaufwendig wäre. Um einen direkten Vergleich durchzuführen, müssten die Anwendungsfälle dieser Arbeit eCASE, eCOS und eTAS mit Hilfe eines anderen KI Systems implementiert und deren Qualität verglichen werden.

Die anderen intelligenten Systeme werden daher nur daraufhin untersucht, wie sie sich bei paralleler Ausführung verhalten und ob dadurch positive Auswirkungen auf die Qualität festgestellt werden können. Inzwischen ist bekannt, dass Computer nicht mehr schneller werden, da die physikalische Grenze erreicht ist. Zukünftige Computer werden daher nur noch an Geschwindigkeit durch Parallelisierung zunehmen. Dies bedeutet, dass intelligente Algorithmen entwickelt werden müssen, die mit erhöhter Parallelisierung automatisch an Qualität gewinnen.

Das ursprüngliche Ziel dieser Arbeit war die Entwicklung eines Kassenleitsystems für Globus. Daher wurden verschiedene Möglichkeiten zur Lösung dieses Problems herangezogen. Es soll geklärt werden, warum ein neues KI-System entwickelt und kein bestehendes System verwendet wurde.

4.1 Automatisches Planen

Automatisches Planen ist ein Konzept, bei dem versucht wird, die Einschränkungen eines Problemlösungsagenten zu überwinden [47]. Planer sind Programme, die entworfen wurden, um eine große Menge an Problemen effizient zu lösen [44]. Probleme werden dazu in einer für den Planer verständlichen Sprache (häufig PDDL) spezifiziert. Der Planer gibt, sobald er eine Lösung findet, diese als eine schrittweise Aktionsliste aus. Es gibt eine Vielzahl an Planern. Um sie zu testen und besser zu klassifizieren, werden sie in einem jährlichen Turnier (die IPC, engl. „International Planning Competition“ [1]) verglichen. Da die Ansätze sehr vielseitig sind, unterscheidet man inzwischen deterministische Planer, lernende Planer, kontinuierlich probabilistische Planer und diskret probabilistische Planer. Diese Kategorien untergliedern sich jeweils noch weiter.

Generell können Planer parallelisiert werden. Dies bezeugt allein die Existenz eines eigenen Bereichs für Mehrkernplaner unter den deterministischen Planern. Zu klären ist, wie sich die Verwendung mehrerer Kerne auf die Qualität auswirkt. Mehr Kerne bieten in erster Linie einen Geschwindigkeitsvorteil. Dadurch kann der Planer natürlich in gleicher Zeit mehr Pläne finden und folglich auch bessere. Planer erzielen also durch Parallelität eine Qualitätssteigerung.

[1] http://ipc.icaps-conference.org/

Möchte man einen Planer benutzten, um eine Entscheidung zu fällen, so muss man dies als Problem konstruieren. Dies ist generell erstmal nicht möglich. Ein Plan liefert eine Möglichkeit, ein Ziel zu erreichen. Bei einer Entscheidung geht es eher darum, auf welche Weise das Ziel erreicht wird. Natürlich kann man eine Entscheidung als Problem modellieren, dennoch ist eine Entscheidung dann eine Auswahl zwischen Plänen, nicht das Generieren eines Plans. Will man es dennoch tun, so müsste die Planqualität die Qualität der Entscheidung widerspiegeln. Zusammengefasst würde man einen Planer eher nicht heranziehen, wenn man eine Entscheidung fällen möchte. Daher scheiden Planer für die Entwicklung eines Kundenleitsystems aus.

Planer für eine Entscheidung zu verwenden hat einen weiteren Nachteil. Es ist schwierig die Laufzeit eines Planers zu bestimmen. Er kann je nach Ausprägung des Problems eine unterschiedlich lange Laufzeit haben. Auch wenn er zu gewissen Problemen häufig Lösungen schnell findet, kann dies nicht garantiert werden [44].

4.2 Künstliche Neuronale Netze

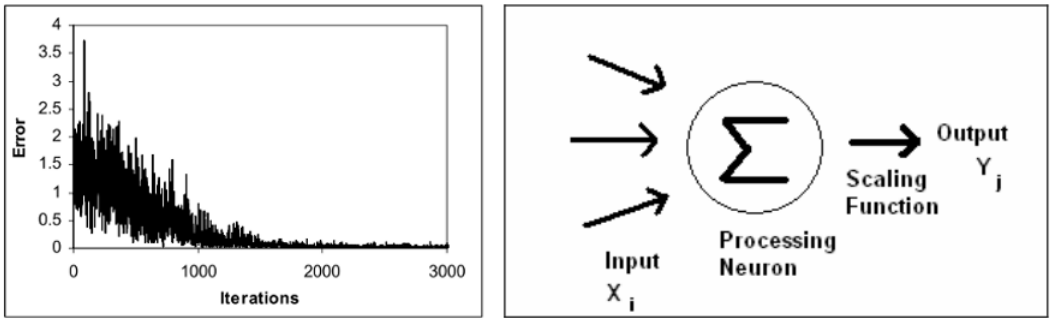


Abbildung 4.1: Darstellung der Fehlerfunktion und des schematischen mathematischen Aufbaus eines künstlichen Neuronalen Netzes [31]

Neuronale Netze werden dem zentralen Nervensystem nachempfunden, bei dem Neuronen durch ein Netzwerk aus Zellen geschickt werden. Dieses Vorbild wurde genutzt, um ein mathematisches Modell zu konstruieren, welches die Vernetzung und Weiterleitung von Neuronen nachbildet [47].

Künstliche Neuronale Netze verknüpfen einen Eingabevektor X_i mit einem Ausgabevektor Y_i anhand von gegebenen Beispielen (vgl. Abbildung 4.1). Diese Verknüpfung kann dann für unbekannte Eingabevektoren benutzt werden, um sie mit entsprechenden Ausgaben zu verknüpfen. Eine bekannte Anwendung der Künstlichen Neuronalen Netze sind Klassifizierer. Sie ordnen Eingaben in verschiedene Klassen ein. Ein binärer Klassifizierer kann zum Beispiel dazu verwendet werden zu erkennen, ob ein Foto ein Gesicht beinhaltet oder nicht. Der Eingabevektor wären die Pixel des Bildes und die Ausgabe eine binäre Klassifikation: Gesicht oder kein Gesicht. Man würde das neuronale Netz mit bekannten Fotos trainieren, um anschließend effizient unbekannte Fotos automatisch zu klassifizieren.

Künstliche Neuronale Netze sind grundsätzlich parallelisierbar [36, 31]. Die Qualität der Erkennung steigt aber nicht durch die parallele Ausführung. Theoretisch kann ein

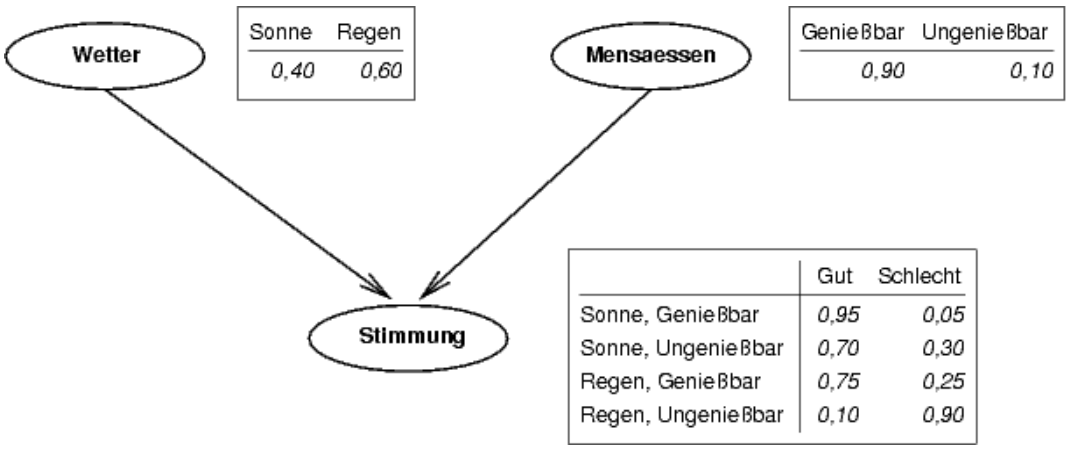


Abbildung 4.2: Beispiel eines Bayesschen Netzes
`http://commons.wikimedia.org/wiki/File:Bayesses_Netz.png`

größeres System auch qualitativ besser arbeiten, aber bei gleicher Eingabe und mit gleichen Trainingsdaten wird die Entscheidung durch mehr Parallelisierung nicht besser.

Künstliche Neuronale Netze werden sehr häufig für Entscheidungsprobleme herangezogen [36, 43, 27]. Sie können also generell für Entscheidungen trainiert werden. Ohne geeignete Trainingsdaten können Neuronale Netze aber nicht trainiert werden. Für das Einkaufsszenario müssten große Mengen an Trainingsdaten generiert werden. Das System wäre danach aber nur so gut, wie die zugrunde liegenden Trainingsdaten. Würden diese von einem Menschen generiert, so wären die Entscheidungen höchstens so gut, wie die eines Menschen mit einer Übersicht über alle Kassen. Würde man einen Algorithmus verwenden, um die Trainingsdaten zu generieren, könnte man diesen gleich für die Entscheidung heranziehen. Systematisch gesehen gleichen die Entscheidungen, die durch Künstliche Neuronale Netze erzeugt werden, den spontanen Entscheidungen. Daher sind Neuronale Netze ungeeignet um eine Kassenempfehlung zu berechnen.

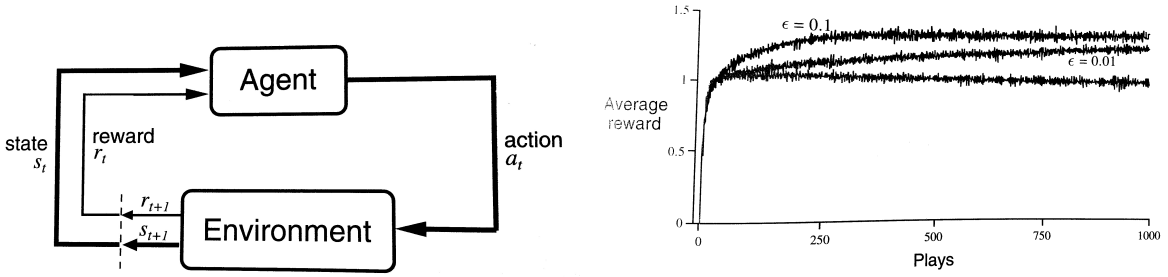
Die Laufzeit künstlicher Neuronaler Netze ist im Wesentlichen unabhängig vom Eingabevektor konstant. Die Auswertung des Netzes hat immer die gleiche Anzahl von Schritten und hängt nur von der Größe des Netzes und üblichen geringen Laufzeitschwankungen ab.

4.3 Bayessche Netze

Ein Bayessches Netz ist ein gerichteter azyklischer Graph, dessen Knoten Zufallsvariablen darstellen, die mit Tabellen annotiert sind, welche die Wahrscheinlichkeitsverteilung der Zufallsvariablen darstellen, und dessen Kanten die bedingten Abhängigkeiten zu anderen Zufallsvariablen anzeigen. Es kann theoretisch jede Verteilung verwendet werden, praktisch werden häufig[2] diskrete Verteilungen oder Normalverteilungen verwendet.

Abbildung 4.2 zeigt ein Beispiel für ein Bayessches Netz. Bayessche Netze werden unter anderem für Induktion, Deduktion, Medizin [59] und Planerkennung [12] verwendet. Dynamische Bayessche Netze sind Bayessche Netze, die sich über die Zeit hinweg

[2]Quelle: `http://de.wikipedia.org/wiki/Bayessches_Netz`, 20. Mai 2014



(a) Lernkreislauf (b) Auswirkungen des Greedy-Faktors

Abbildung 4.3: Bestärkendes Lernen nach Sutton und Barto [52]

verändern. Sie können zur Einschätzung von Zuständen, in denen sich Menschen befinden, verwendet werden [28]. Die vier möglichen Begründungsvarianten (vgl. Abbildung 8.3), die man in einem Bayesschen Netz verwenden kann sind: diagnostisch, vorhersagend, interkausal, und kombiniert. Zur Berechnung konkreter Wahrscheinlichkeiten im Bayesschen Netz verwendet man die folgende Formel

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | Eltern(X_i))$$

Es ist möglich, mit einem Bayesschen Netz Entscheidungen zu fällen. Diese sind aber zustandsorientiert, das bedeutet mit Hilfe des Bayesschen Netzes wird ein Zustand bestimmt (bzw. dessen Wahrscheinlichkeit) und damit eine Entscheidung gefällt. Entscheidungen, die man für eine Ampel oder für eine Kasseneinschätzung fällt, sind nicht zustandsorientiert. Für eine Kasse könnte man versuchen die folgenden Zustände zu inferieren: Voll, Frei, Belegt, hohe Wartezeit, geringe Wartezeit. Diese folgern sich direkt aus der Wartezeit und daher wären Bayessche Netze ein Umweg.

Bayessche Netze können parallelisiert und verteilt werden [40]. Dies verbessert die Qualität nur in dem Sinne, dass größere Netze effizient berechnet werden können. Bei gleicher Datenbasis findet keine Verbesserung statt.

Die Laufzeit einer Anfrage an ein Bayessches Netz berechnet sich immer aus der gleichen Formel. Daher ist die Laufzeit sehr gut vorhersagbar. Sie variiert durch Veränderungen am Netz geringfügig. Das Hinzufügen neuer Abhängigkeiten führt zu einer erhöhten Laufzeit, da hier bei einer erneuten Berechnung ein weiterer Zweig im Netz einbezogen werden muss. Das Hinzufügen von Abhängigkeiten wird typischer Weise nicht zur Laufzeit durchgeführt (Ausnahme: Dynamische Bayessche Netze).

4.4 Bestärkendes Lernen

Bestärkendes Lernen (auch Verstärkendes Lernen) klassifiziert eine Gruppe von Algorithmen aus dem Bereich des maschinellen Lernens. Bestärkendes Lernen ordnet möglichen Aktionen in einem Zustand automatisch einen Nutzen zu und verteilt diesen so über alle Zustände und Aktionen, sodass der Nutzen von Aktionen vorhergesagt werden kann. Bestärkendes Lernen basiert auf den Markow-Entscheidungsprozessen (engl. Markov Decision Processes, MDP). Um zu entscheiden, welche Aktion der Agent im

aktuellen Zustand ausführt, versucht er den erwarteten Gewinn zu maximieren. Um den Nutzen neuer Aktionen zu lernen, führt der Agent mit einer gewissen Wahrscheinlichkeit (Greedy-Faktor ϵ) eine berechnet optimale Aktion durch. Ansonsten nimmt er eine zufällige Aktion und aktualisiert den Nutzen später durch die erwirtschaftete Resonanz. Dieser Greedy-Faktor entscheidet also, wie oft der Agent versucht, Neues zu lernen und wie oft er Gelerntes ausnutzt [52]. Der Erfolg von Bestärkendem Lernen spiegelt sich in seiner Vielzahl von Anwendungen wider[3], wie zum Beispiel autonome Helikoptersteuerung [42] oder beim Roboterfußball [51].

Bestärkendes Lernen kann dazu verwendet werden, Entscheidungen zu fällen [26]. Dazu wird die Entscheidung einem virtuellen Agenten überlassen und diese dann in der Realität umgesetzt. Durch die Verteilung des Lernprozesses auf mehrere unabhängige Agenten kann eine bessere Qualität der Entscheidungen einzelner Agenten erzielt werden [7]. Die Laufzeit von Bestärkendem Lernen kann sehr gut vorhergesagt werden, da sie sich auf eine einfache mathematische Formel reduzieren lässt, deren Auswertung allein von der Anzahl der Zustände und möglichen Aktionen abhängt.

Bestärkendes Lernen kann beim Kassenverhalten jedoch nicht sinnvoll eingesetzt werden. Das Problem beim Bestärkenden Lernen ist die Resonanz. Würde man ein Kassenleitsystem mit Bestärkendem Lernen implementieren, so müsste man den Kunden die vorgegebene Entscheidung bewerten lassen. Dies würde erfordern, dass der Kunde seine Situation richtig einschätzt und das System richtig bewertet. Dies ist aber unrealistisch, da der Kunde keinen Überblick über die gesamte Situation hat. Würde man das System mit einem Greedy-Faktor von 1, also ohne Erkundung neuer Aktionen implementieren, so müsste man das System im voraus trainieren. Dies ist grundsätzlich möglich, würde aber die Anpassungsfähigkeit auf neue Situationen im Laden zunichte machen. Das Hauptproblem ist also einen Ablauf zu spezifizieren, bei dem eine geeignete Resonanz für die Entscheidung an das System zurückgegeben wird.

4.5 Statische Algorithmen

Statische Algorithmen sind Algorithmen, die sich entsprechend einer vom Programmierer implementierten Logik verhalten. Dazu muss der Programmierer eine konkrete Problemlösung finden und diese umsetzen. Ein Beispiel für einen statischen Algorithmus bei einer Kassenwahlunterstützung wäre eine Gleichverteilung auf alle Kassen oder immer die Kasse mit der geringsten Anzahl an wartenden Personen zu empfehlen.

Statische Algorithmen können gut zur Entscheidungsfindung genutzt werden. Sie sind je nach Implementierung auch durch Parallelisierung optimierbar. Über die Laufzeit können keine Aussagen getätigt werden.

Für die Kassenwahl stellt sich heraus, dass ein Hinweisen auf die Kasse mit den wenigsten wartenden Personen statistisch nicht von der Bewertung durch das hier vorgestellte KI-System unterscheidet. Dies gilt, obwohl dieser statische Algorithmus einige spezielle Fälle schlechter als die KI unterscheidet, diese jedoch so selten auftreten, dass sie nicht ins Gewicht fallen.

Im Nachhinein stellte sich heraus, dass ein statischer Algorithmus für die Kassenwahl der geeignete Algorithmus gewesen wäre, da er das Problem fast genau so gut,

[3] `http://umichrl.pbworks.com/w/page/7597597/Successes%20of%20Reinforcement%20Learning` Zusammenfassung der Erfolge von Bestärkendem Lernen

jedoch wesentlich effizienter löst. Beim Verkehrssystem ist dies jedoch nicht der Fall. Dies bestätigt, dass die KI das Problem der Kassenwahl nicht besser lösen kann, da es möglicher Weise keine bessere Lösung für das Problem gibt (Mit Ausnahme des konstruierten Spezialfalls siehe Evaluation).

4.6 Zusammenfassung

Tabelle 4.1: Übersicht und Vergleich der Techniken
* Ist Implementierungsabhängig

Technik	Entscheidungen	Parallelisierbar	Adaptiv	Bessere Qualität durch Parallelisierung	Laufzeit vorhersagbar
Automatisches Planen	Nein	Ja	Ja	Ja	Nein
Neuronale Netze	Ja	Ja	*	Nein	Ja
Bayessche Netze	Nein	Ja	*	Nein	Ja
Bestärkendes Lernen	Ja	Ja	Ja	Nein	Ja
Statische Algorithmen	Ja	*	Nein	*	*
Trajektoriensimulation	Ja	Ja	Ja	Ja	Ja

Die Trajektoriensimulation ist geeignet, um fehlerhafte Sensordaten auszugleichen. Durch geringe Fehler bei der Wahrnehmung kommt es auch nur zu einem geringen Fehler bei der Approximation des Erwartungswerts. Eine Fehlentscheidung kommt also zustande, wenn die durch Fehler verursachte Verschiebung des Erwartungswerts größer ist als der Abstand der Erwartungswerte. Dies sollte bei geringen Verschiebungen nur eintreten, wenn der Abstand auch gering ist. Unterscheidet sich der Erwartungswert nur gering, ist die Auswirkung der Entscheidung entsprechend gering. Daher ist die Trajektoriensimulation fehlertolerant.

Die anderen Algorithmen, die sich zur Entscheidungsfindung eignen, müssen mit Trainingsdaten generiert werden. Die Neuronalen Netze orientieren sich daher an der Entscheidungsfähigkeit eines Menschen, der die Trainingsdaten auswertet. Bestärkendes Lernen benötigt um adaptiv zu bleiben ein kontinuierliches Feedback, das in einem Kundenleitsystem schwer umsetzbar ist. Trajektoriensimulation benötigt lediglich ein Verständnis des Ablaufs und muss diesen in der Form eines Simulators nachbilden. Dadurch gibt man dem Computer ein allgemeines Verständnis der Situation und ermöglicht es, Alternativen zu durchdenken.

Die Tabelle 4.1 zeigt eine Übersicht der vorher beschriebenen Techniken im Vergleich zu der hier vorgestellten Trajektoriensimulation. Es zeigt sich, dass die Umsetzung der Trajektoriensimulation eine sinnvolle Erweiterung des Spektrums der existierenden KI-Systeme ist.

5 Implementierung

5.1 Rahmenwerk für Trajektoriensimulationen

Das „Trajectory-Simulation-Decision-Framework“ (kurz TSDF) ist ein Rahmenwerk, welches die Entwicklung von intelligenten Anwendungen vereinfacht, die Echtzeitentscheidungen über Trajektoriensimulationen in Dual Reality verwenden sollen. Das TSDF Rahmenwerk soll Entscheidungen autonom fällen können. Es soll dabei unabhängig von der instrumentalisierten Umgebung arbeiten. Dazu soll im Folgenden ein Software-Entwicklungsprozess angestoßen werden, damit das Rahmenwerk dokumentiert, wiederverwendbar und flexibel gehalten wird. Zuerst sollen jedoch einige entwicklungsspezifische Fragen behandelt werden, die in einem normalen Softwareprozess erst später geklärt werden, hier jedoch vorgezogen werden müssen, da sie bereits vor den Anforderungen spezifiziert waren.

- **Warum ein Rahmenwerk?**
  Ein Rahmenwerk unterscheidet sich von einer Bibliothek dadurch, dass es Klassen und Strukturen hat, die abgeleitet und nicht nur Funktionen, die aufgerufen werden. Dadurch wird im wesentlichen die Programmiersprache festgelegt, denn ein Rahmenwerk kann nicht ohne weiteres in eine andere Programmiersprache portiert werden, wogegen eine Bibliothek Fremdverwendung durchaus zulassen kann. Zur Simulation wird eine konkrete Schnittstelle benötigt, da die genaue Implementierung eines Simulators für die Entscheidung irrelevant ist, gewisse Funktionalitäten aber bereit gestellt werden müssen. Für die Evaluation eines Models, sowie die Ausführung der Entscheidung werden gewisse Informationen über die Umgebung benötigt, die nicht über funktionale Aufrufe an eine Bibliothek übergeben werden können. Daher bietet es sich an, ein Rahmenwerk als Implementierungsvariante zu wählen. Die Frage nach einer Anwendung im Gegensatz zur Bibliothek oder Rahmenwerk stellt sich nicht, da die entwickelte Software nicht ausführbar sein soll.

- **Warum Java?**
  Die Entscheidung der Programmiersprache wurde nicht, wie üblich, im Entwicklungsprozess geklärt, sondern ergab sich aus einer indirekten Vorgabe für eine zukünftige Integration im IRL oder im Globus. Obwohl es nicht als Teil der Arbeit spezifiziert ist, soll das Rahmenwerk zukünftig im IRL oder vielleicht in einem realen Supermarkt verwendet werden. Daher liegt es nahe, die bekannte Infrastruktur des IRLs zu verwenden, die nahezu komplett auf Java basiert. Ferner ist Java eine in der Wissenschaft weit verbreitete und bekannte Sprache, was die Verwendung ebenfalls nahe legt.

- **Warum der EBS als Infrastruktur?**
  Die wichtigste Abhängigkeit von TSDF, die später näher erläutert wird, ist der

„Event Broadcasting Service“, eine für Smart Environments entwickelte Kommunikationsinfrastruktur, die es ermöglicht, effizient Daten zwischen Sensoren, Aktuatoren und Services auszutauschen, ohne sich um Feinheiten wie ein Protokoll oder Verbindungen zu kümmern [19, 17, 18].

Die *EBSLib* beinhaltet diesen Service und ist die im IRL verwendete Infrastruktur. Daher ist es sinnvoll, dass TSDF an den EBS anzuschließen. So können leicht echte Sensoren und Aktuatoren eingebunden werden.

5.1.1 Anforderungsanalyse

Das TSDF soll es ermöglichen Anwendungen zu schreiben, die auf eingehende Events reagieren und es darauf basierend ermöglichen, Entscheidungen autonom zu fällen, in dem es Trajektorien simuliert und diese nach spezifizierten Bewertungsfunktionen auswertet und die beste Alternative anwendet. Dabei sollen mehr durchgeführte Simulationen statistisch zu besseren Ergebnissen führen.

Anforderungen

- Es soll möglich sein, dass die Entscheidungen auf einem durch das Netzwerk verbundenen Computer durchgeführt werden, um die Ressourcen besser zu verteilen.
- Die Simulationen müssen in eigenen *Threads* durchführbar sein, um die heute üblichen Mehrkernprozessoren besser ausnutzen zu können.
- Sensoren und Aktuatoren sind ausschließlich über das Netzwerk erreichbar. Zur Kommunikation wird die eventbasierte Kommunikationsinfrastruktur EBS verwendet.
- Das Rahmenwerk muss Entscheidungen auch mit unzureichenden oder von der Realität abweichenden Sensordaten fällen können. Dabei soll die Laufzeit einer Entscheidung weitestgehend konstant sein.
- Haben zwei oder mehrere Alternativen die gleiche Bewertung, so wird die in der Reihenfolge zuerst auftretende Alternative zurückgegeben. Dadurch kann die Verbindung zu genetischen Algorithmen besser umgesetzt werden, da die aktuelle Situation als erste Alternative spezifiziert werden kann und diese so durch ihre Reihenfolge bei Gleichstand einen Vorteil hat.

5.1.2 Spezifikation der Schnittstelle

Abbildung 5.1 zeigt ein vereinfachtes UML Diagramm des TSDF. Die Entscheidungen werden von der Klasse *DecisionMaker* durchgeführt, sie bekommt dazu mehrere Alternativen der Klasse *Case* und führt diese aus. Jeder *Case* erzeugt für die Anzahl der durchzuführenden Simulationen Kopien des *Simulators* und gibt diese an einen jeweils eigenen *CaseThread* zur Ausführung und Bewertung weiter. Der *CaseThread* führt dann die Simulation durch und wendet zum Zeitpunkt x, spezifiziert durch *getApplyTime*, die in der Methode *apply* von *Case* aus, um die spezifizierte Änderung am Modell des Simulators durchzuführen. Am Ende ruft der *CaseThread* die Methode *evaluate* des

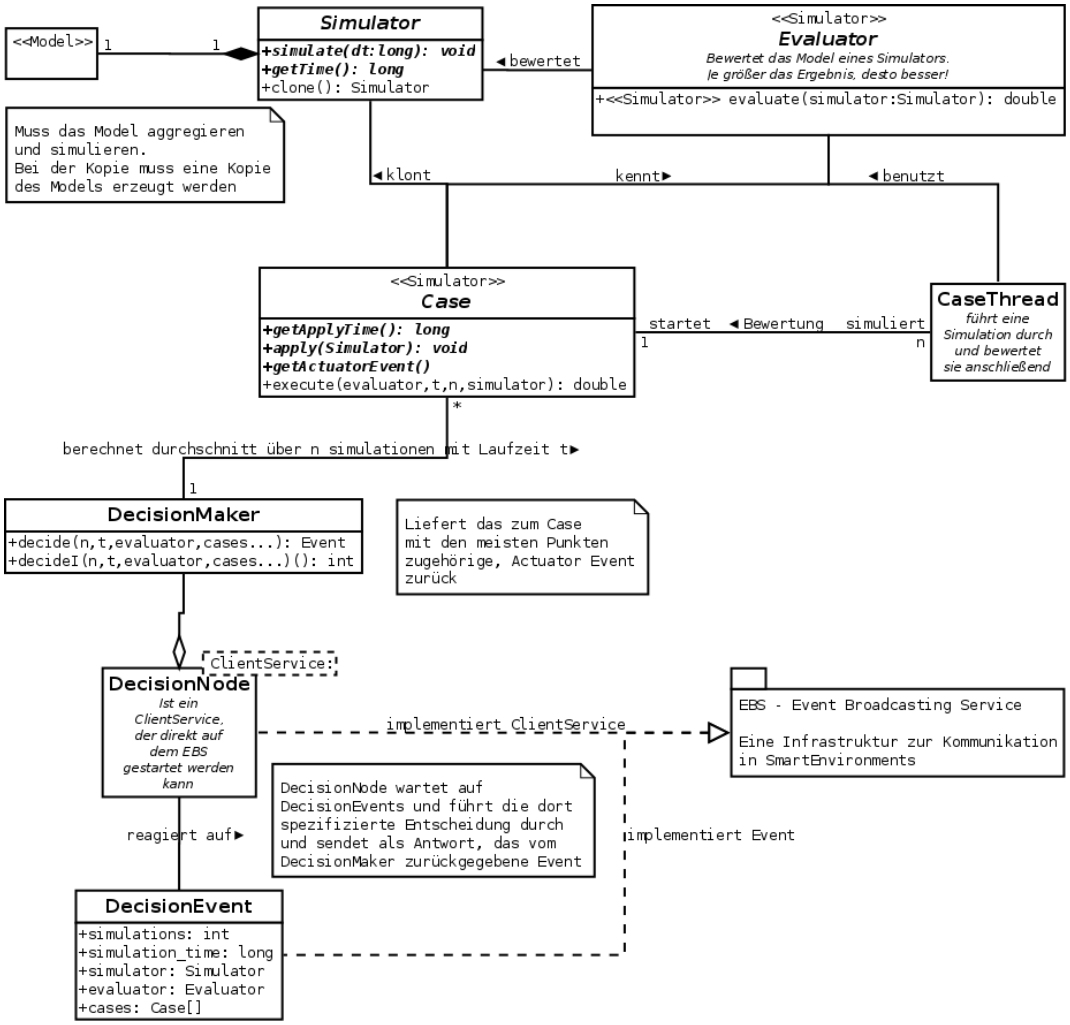


Abbildung 5.1: UML Diagramm des Trajectory Simulation Decision Frameworks (TSDF) zur Darstellung der Schnittstelle

Evaluators auf, um die Simulation zu bewerten. Der *Case* bildet nach der Beendung aller *CaseThreads* die durchschnittliche Punktzahl und gibt diese an den *DecisionMaker* zurück. Dieser liefert dann das *ActuatorEvent* der Alternative mit der höchsten durchschnittlichen Punktzahl zurück und fällt so die Entscheidung.

Für eine Entscheidung benötigt das Rahmenwerk eine Implementierung des *Simulators*, mindestens zwei Implementierungen einer Alternative *Case* und eine Implementierung der Klasse *Evaluator*.

Beschreibung der Aufgaben und Funktionen der Klassen

- **Simulator**
  Die Klasse *Simulator* aggregiert ein Modell, welches eine virtuelle Darstellung der Wirklichkeit wie im Konzept beschrieben enthält. Diese simulierte Wirklichkeit folgt einer eigenen diskreten Uhr, die in einer beliebigen aber festen Einheit über die Methode *getTime* zwischen den Aufrufen von *simulate* abgerufen werden kann und die aktuelle Zeit des *Simulators* zurück gibt. Die Methode *simulate* nimmt eine beliebige Zeitdifferenz *dt* und simuliert die Wirklichkeit entsprechend in die

Zukunft. Folglich muss sich *getTime* vor und nach dem Aufruf von *simulate* um exakt *dt* Zeitschritte unterscheiden. Die Methode *clone* ist fertig implementiert und liefert eine tiefe Kopie der Instanz zurück. Sie kann überschrieben werden, sofern es eine effizientere Variante gibt, um die Klasse tief zu kopieren. Die Methode *simulate* sollte darauf konzipiert sein, kleine Zeitdifferenzen zu simulieren und dabei möglichst geringe Fehler zu machen. Die Simulation muss so effizient gestaltet sein, dass die später zur Verfügung gestellte Zeit ausreicht, um das dann spezifizierte Intervall zu simulieren. Wenn nicht, wird die Simulation abgebrochen und frühzeitig evaluiert.

- **Evaluator**
  Der *Evaluator* ist eine Implementierung einer Bewertungsfunktion. Sie bekommt als Eingabe einen *Simulator* und kann von diesem das Modell beliebig bewerten. Die Bewertung sollte bei gleicher Eingabe die gleiche Ausgabe machen. Man kann aber auch zufällige Bewertungen zurückgeben. Es gilt: Je größer die zurück gegebene Zahl ist, desto besser wird die Simulation und das zugehörige Modell bewertet. Negative Zahlen sind explizit erlaubt. Die Verwendung der Werte *NaN* (Not a Number) und *Infinity* müssen vermieden werden!

- **Case**
  Ein *Case* spiegelt im Rahmenwerk eine mögliche Alternative einer Entscheidung wider. Die Klasse *DecisionMaker* erwartet bei einer Entscheidung mehrere Instanzen von einer oder verschiedener *Case*-Implementierungen. Ein *Case* spezifiziert eine Zeit, nach der die Methode *apply* vom Rahmenwerk aufgerufen wird. Ein *Case* folgt immer dem folgenden Ablauf:

  1. Klone k Simulatoren
  2. Rufe für jeden Simulator einen *CaseThread* auf mit $t = x + y$ Zeitschritten.
     a) Simuliere die Kopie $x =$ *case.getApplyTime()* Zeitschritte
     b) Rufe *apply(copy)* auf
     c) Simuliere bis zum Zeitpunkt y ($t -$ *applytime* Zeitschritte).
     d) Bewerte die Kopie
     e) Liefere die Bewertung zurück
  3. Warte auf die Ergebnisse aller *CaseThreads*
  4. Liefere den Durchschnitt zurück

- **DecisionMaker**
  Möchte man keine *DecisionNode* verwenden um Entscheidungen in einem Netzwerk durchzuführen, so muss man eine Instanz von *DecisionMaker* erzeugen und die Methode *decideI* aufrufen, um eine Entscheidung zu fällen. Der zurückgegebene Index zeigt auf den *Case* mit der höchsten Punktezahl. Alternativ kann mit *decide* das entsprechende Event zurückgegeben werden. Bei gleicher Punktzahl entscheidet die Reihenfolge.

  DecisionMaker.decide benötigt folgende Parameter:

  - *simulations*: Die Anzahl k der Simulationen, die pro *Case* durchgeführt werden sollen.
  - *simulation time*: Die Zeitschritte, die insgesamt für die Simulation verwendet werden sollen, in Zeitschritten des Simulators $(x + y)$
  - *sim*: Die Implementierung einer Simulatorklasse
  - *evaluator*: eine Implementierung der Evaluatorklasse, passend zu dem entsprechenden Simulator
  - *cases*: Ein Array von *Case*-Instanzen, die als Alternativen der Entscheidung mit der Anwendungszeit x verwendet werden

- **DecisionNode**
  Die *DecisionNode* ist eine Implementierung der Klasse *ClientService* aus der *EBS-Lib*. Eine *DecisionNode* kann auf jedem *PhysicalDeviceService* gestartet werden und wartet dort auf ein *DecisionEvent*. Wird ein *DecisionEvent* empfangen, wird die darin spezifizierte Entscheidung gefällt und das als Ergebnis zurückgegebene Event in das Netzwerk geschickt.
  Dadurch ist es möglich, Entscheidungen auf andere Computer im Netzwerk zu übertragen. Wird dort eine *DecisionNode* gestartet, so kann die Entscheidung an diesen Computer delegiert werden. Das Ergebnis wird direkt an den Aktuator weitergeleitet. Betreibt man mehrere *DecisionNodes* in einem Netzwerk, so muss man im Event *setPrivateFlag* aufrufen und das Event adressiert übermitteln, da sonst der *EBS* das Event an alle *DecisionNodes* schickt und diese dann die gleiche Entscheidung berechnen und jeweils eine Antwort abschicken. Dies könnte dazu führen, dass verschiedene Alternativen als Lösung für die gleiche Entscheidung umgesetzt werden. Betreibt man nur eine *DecisionNode* kann das Event auch als *Broadcast* übermittelt werden. Generell ist davon jedoch abzuraten, da ein *DecisionEvent* eine komplette Serialisierung eines Simulators beinhaltet, welcher zusammen mit einem Modell recht groß sein kann und alle im Netzwerk vorhandenen *ClientServices* diese Nachricht entpacken müssten.

- **DecisionEvent**
  Ein *DecisionEvent* spezifiziert die Parameter, die ein *DecisionMaker* benötigt, um eine Entscheidung zu fällen. Die exakte Beschreibung der einzelnen Attribute kann dessen Beschreibung entnommen werden. *DecisionEvents* sollten immer als adressiertes Event mit gesetztem *Private-Flag* im *EBS* verschickt werden, damit sie nicht als *Broadcast* versendet werden.

5.1.3 Abhängigkeiten

```
<dependency>
  <groupId>de.dfki.irl</groupId>
  <artifactId>EBSLib</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency><dependency>
  <groupId>uk.com.robust-it</groupId>
  <artifactId>cloning</artifactId>
  <version>1.9.0</version>
</dependency>
```

Listing 5.1: Maven Dependencies

Das TSDF benötigt zwei Abhängigkeiten. Zum einen die *EBSLib* vom IRL und zum anderen eine effiziente Bibliothek zum tiefen Klonen von Instanzen *cloning* von „rubust-it“ (vgl. Listings 5.1). Die Abhängigkeit *cloning* ist im Central-Maven-Repository verfügbar, die *EBSLib* kann nur im DFKI intern automatisch abgerufen werden, steht aber seit der Veröffentlichung von Gerrit Kahl [19] zur Verfügung. Die Implementierung der *EBSLib* wurde dieser Arbeit beigefügt und kann manuell in das lokale Repository installiert werden. Die *EBSLib* besitzt einige weitere Abhängigkeiten, die jedoch alle aus dem Central-Maven-Repository bezogen werden können.

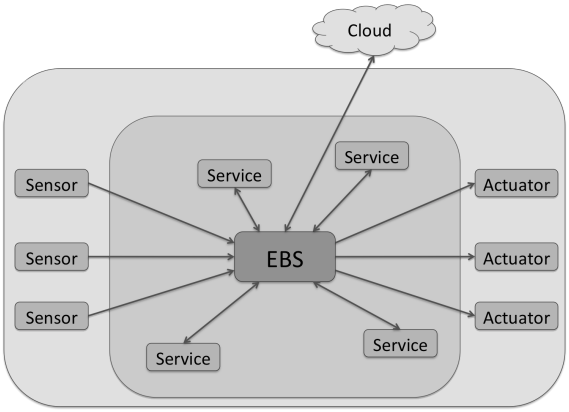


Abbildung 5.2: Architektur des EBS[18]

Die *EBSLib* ist ein Rahmenwerk zur einfachen Verknüpfung einer instrumentalisierten Umgebung und kann optimal zur Synchronisation einer Dual Reality verwendet werden. Die *EBSLib* stellt einen *BroadcastServer* zur Verfügung, zu dem sich verschiedene Services verbinden können. Diese Services können Events empfangen und versenden. Das Grundprinzip der Kommunikationsinfrastruktur ist, dass alle verbundenen Services alle Events bekommen. Man kann auch adressierte Events verschicken, die nur dem spezifizierten Ziel zugestellt werden. Außerdem können die Events verschlüsselt oder offen übertragen werden. Abbildung 5.2 zeigt die schematische Darstellung eines EBS. Sensoren und Aktuatoren werden mit Services verknüpft und ermöglichen so eine schnelle und flexible Implementierung einer Dual Reality.

Die Simulierte Realität, Virtuelle Realität und Hypothetische Realität werden in dieser Arbeit als Services eines EBS implementiert. Dadurch kann die Simulierte Realität virtuelle Sensoren und Aktuatoren implementieren, die später leicht durch echte ersetzt werden können. Die Virtuelle Realität, die für die Entscheidungen verantwortlich ist, muss dazu nur geringfügig angepasst werden. Die vorhandenen Sensoren und Aktuatoren müssen natürlich die entsprechenden EBS Schnittstellen implementieren.

5.2 Ampelsteuerung

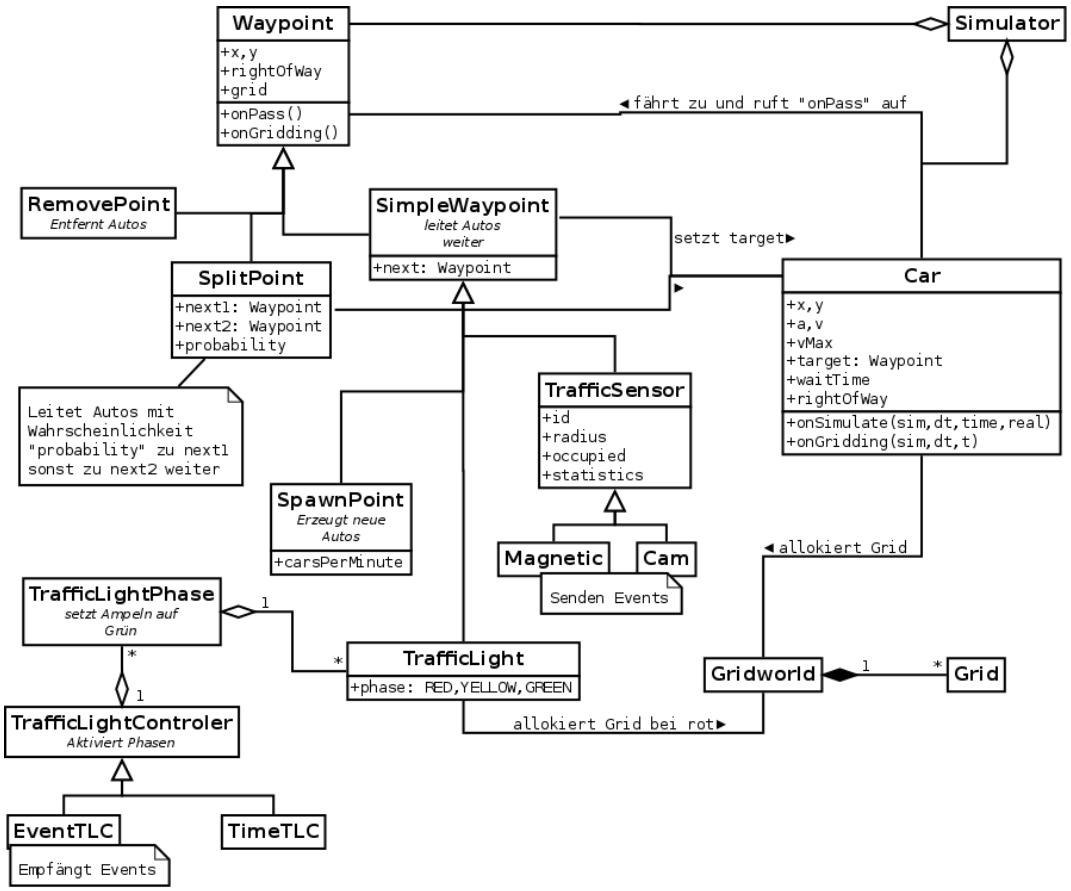


Abbildung 5.3: Stark abstrahiertes UML Klassenmodell des Ampelsystems

Die Abbildung 8.1 zeigt eine geschnittene Grafik der grafischen Oberfläche bei einer laufenden Simulierten Realität. Autos sind als Objekte mit einer Position, einer Geschwindigkeit und einer Beschleunigung modelliert. Autos fahren immer auf dem direkten Weg zur Position ihres Ziels und sobald sie dieses erreicht haben, rufen sie dort die Methode *onPass* auf. Die Logik der Positionen wird durch die Wegpunkte implementiert. Dadurch folgen die Autos einer Spur, die als eine Folge von Wegpunkten definiert ist.

Das Listing 5.2 zeigt, wie die Logik der Spuren implementiert wird. Zuerst wird der *seed* des Pseudozufallszahlengenerators (kurz PRG engl. Pseudo Random Generator) auf die Autonummer, Kreuzungsposition und die Zeit des Autos gestellt. Dies stellt sicher, dass das selbe Auto sich an der selben Kreuzung immer gleich entscheidet. Dies wird benötigt, da für die Kollisionslogik mehrere Kopien der Welt erzeugt werden und sich dort die Autos gleich entscheiden müssen. Anschließend wird das nächste Ziel des Autos, entsprechend der Ausgabe des PRG, festgelegt.

```
    @Override
    public void onPass(TrafficSimulator world, Car car, boolean real){
        super.onPass(world, car, real);

        //Der Weg eines Autos entscheidet sich nach seiner
        //id, Position und Zeit.
        random.setSeed(car.getCarId() * x * y * car.getSpawnTime());

        //Ziel des Autos setzen
        if (random.nextFloat() * 100 <= probability) {
            car.setTarget(next1);
        } else {
            car.setTarget(next2);
        }
    }
```

Listing 5.2: Implementierung der onPass Methode des SplitPoint

Die Implementierung verwendet ein Raster, um eine Kollisionsprüfung durchzuführen. Das Raster wird ebenfalls von Ampeln blockiert, sofern diese rot sind. Listing 5.3 zeigt, wie eine Ampel eine Rasterposition alloziert. Je nach Phase passiert etwas unterschiedliches: Ist die Ampel grün, orangerot oder ausgeschaltet (blinking), so werden alle Rasterpositionen freigegeben. Ist die Ampel jedoch rot, so wird die Rasterposition permanent blockiert. Ist die Ampel orange, so versucht sie die Rasterposition zu blockieren, schafft dies aber nur, wenn sie nicht von einem Auto belegt ist. Im Gegensatz zu einem Auto blockiert eine Ampel höchstens eine Rasterposition. Dabei wird der Rasterposition keine Zeit der Belegung mitgeteilt, wie dies ein überfahrendes Auto tun würde. Eine rote Ampel hat also einen Vorrang und durch die Kollisionslogik halten die Autos vor einer roten Ampel an.

```
    @Override
    public void onGridding(TrafficSimulator world, double dt, long time) {
        switch (phase) {
            case GREEN:
            case BLINKING:
            case ORANGERED:
                break; //coming from red so please disallocate
            case RED:
                world.getSingleGrid(x, y).permablock(time, this);
                break;
            case ORANGE:
                if (!world.getSingleGrid(x, y).isAllocated(time)) {
                    world.getSingleGrid(x, y).permablock(time, x);
                }
        }
    }
```

Listing 5.3: Implementierung der Methode onGridding eines TrafficLights

5.2.1 Kollisionslogik

Die Kollisionslogik wird wie folgt implementiert:

```
1   foreach (Ampel)
2       Ampel.onGridding()
3
4   sort Cars to RightOfWay,SpawnTime
5
6   foreach (Car)
7       Copy Car
8       Copy.a = MIN_A //volle Bremsung
9       while (Copy.a < MAX_A) //solange nicht volle Beschleunigung
10          Blockiere mit Copy.a erreichbare Grids
11          if possible then Copy.a++
12          else break
13      end
14      Car.a = Copy.a
15  end
```

Listing 5.4: Pseudocode einer Kollisionslogik

Generell berechnet der Simulator nur die Beschleunigung eines Fahrzeugs, um realistische Anfahrtszeiten und Bremswege zur simulieren. Dazu werden zuerst alle durch rote Ampeln markierten Rasterpositionen blockiert. Dann werden die Autos nach aktuellem Vorfahrtrecht und Erscheinungszeit sortiert. Da Überholen unmöglich ist, werden so die Autos auf den Streckenabschnitten von vorne nach hinten iteriert und daher darf das vorderste Auto mit höchstem Vorfahrtrecht zuerst Rasterpositionen blockieren. Da eine Rasterposition zu einer gewissen Zeit nur einmal blockiert werden darf, bewirkt dies ein übliches Vorfahrtrecht ohne Kollision. Kommt es dennoch zu einer Kollision, wird die Geschwindigkeit, entgegen der physikalischen Gesetze, direkt auf Null gesetzt um dies zu verhindern. Steht ein Auto zu lange, da das Antikollisionssystem sich blockiert hat, dürfen die Autos Kollisionen ignorieren und durcheinander durchfahren oder werden entfernt. Beides passiert nur, wenn die Ampel bei viel Verkehr über einen längeren Zeitraum abgeschaltet wird. Dies ist zu erwarten, da ein derart auf Effizienz getrimmtes Antikollisionssystem nicht mit der Fahrintelligenz eines echten Menschen mithalten kann. In den durchgeführten Evaluationen ist dies jedoch nicht eingetreten. Der Fall, dass die Geschwindigkeit direkt auf Null gesetzt wurde, tritt jedoch öfter ein.

Um eine Kreuzung zu erzeugen, können Wegpunkte beliebig instantiiert und verknüpft werden. Dadurch lassen sich viele Straßenszenarien einfach konstruieren und testen. Man könnte die Straßenkonstruktion auch in eine Datei auslagern, da aber zum Testen der KI nur eine Kreuzung konstruiert wurde, ist der Aufbau direkt im Quellcode verankert.

5.2.2 Synchronisation

Die Synchronisation wird durch *CarDetectionEvents* erreicht. Da auf jeder Spur ein eigener Sensor befindet, können Autos zuverlässig erkannt werden. Autos verlassen ihre Spur nicht, daher reicht es, die Magnetresonanzschleifen auszuwerten. *CarPositionEvents* wie sie von Kameras versendet werden, führen lediglich zu einer neuen Positionierung eines Autos und können in diesem Szenario, wie die Ergebnisse später zeigen, völlig ignoriert werden. Wird ein Auto von einem Sensor erkannt, so wird es in der Virtuellen Realität am entsprechenden Sensor eingefügt.

```
1    for (CarDetectionEvent cde : cdes) {
2        InductionSensor sensor =
3            simulator.getSensor(cde.getDetector());
4        Car car = new Car(simulator.getTime(), cde.getX(),
5            cde.getY(), Const.CAR_MAX_V);
6        car.setV(cde.getV());
7        sensor.onPass(simulator, car, true)
8    }
```

Listing 5.5: Synchronisation neuer Autos

Ein weiterer wichtiger Punkt ist die Zeitsynchronisation. Da die Simulierte Realität mit unterschiedlichen Geschwindigkeiten laufen und die Geschwindigkeit der Simulation zur Laufzeit angepasst werden kann, sendet die Simulierte Realität Zeitevents anstatt eine Simulation durchzuführen. Ein Zeitevent beinhaltet einen Zeitpunkt, der von den Simulatoren erreicht werden soll. Empfangen Simulierte Realität, Hypothetische Realität und Virtuelle Realität ein solches Zeitevent, so simulieren sie den Zeitunterschied zu dem im Zeitevent spezifizierten Zeitpunkt. Dadurch ist die Zeit in allen drei Realitäten gleich.

Sowohl die Virtuelle Realität als auch die Simulierte Realität reagieren auf die Events, die durch eine Entscheidung produziert wurden. Dies sind sogenannte *TrafficLightControlEvents* die eine einheitlich gewählte Phasennummer propagieren. Die *TrafficLightController* schalten, sobald ein solches Event empfangen wurde, alle Ampeln zuerst auf rot und dann die der Phase zugeordneten Ampeln auf grün. Dazu wurden in der Simulierten Realität und der Virtuellen Realität gleiche Intervalle für das Umschalten spezifiziert. Da die Phasen nicht an die Zeiten der Simulatoren gebunden sind, kann es zu einer leichten Verzögerung kommen.

5.2.3 Ampelsteuerung

Die Ampeln sollen durch die KI gesteuert werden. Dazu wird im gleichen Netzwerk zusätzlich zu den Realitäten eine *DecisionNode* installiert. Diese reagiert, wie spezifiziert auf *DecisionEvents*. In regelmäßigen Abständen sendet die Virtuelle Realität diese *DecisionEvents* wie folgt:

```
1
2  TrafficLightPhaseCase phase0 = new TrafficLightPhaseCase(0, 0);
3  TrafficLightPhaseCase phase1 = new TrafficLightPhaseCase(0, 1);
4  TrafficLightPhaseCase phase2 = new TrafficLightPhaseCase(0, 2);
5
6  //...
7
8  if (nextdecision < simulator.getTime()) {
9     TrafficSimulator futureSim = (TrafficSimulator) simulator.clone();
10    for (InductionSensor sensor : futureSim.getSensors()) {
11        sensor.setSpawnMode(true);
12    }
13    send(new DecissionEvent(5, 10000, futureSim, new SpeedWaitEvaluator(),
14         phase0, phase1, phase2));
15    nextdecision = nextdecision + decisionInterval;
16 }
```

Listing 5.6: Aufrufen einer Entscheidung im TSDF

Die Ampel besitzt drei Phasen. Für jede Phase wurde ein *TrafficLightPhaseCase* erstellt, der die Phasenkontrolleinheit im Netzwerk spezifiziert (0) und dort eine bekannte Phase setzt $(0, 1, 2)$. In einem festen Intervall wird der aktuelle Simulator der Virtuellen Realität kopiert und mit einem *DecisionEvent* an die *DecisionNode* übermittelt. Dabei werden 5 Simulationen mit jeweils 10.000 Simulationsschritten in Auftrag gegeben. Im Simulator werden die Induktionssensoren noch in den „SpawnMode“ versetzt. Dies bedeutet, dass sie in der Kopie des Simulators Autos entsprechend der zuletzt erkannten Rate erzeugen, anstatt diese zu erkennen. Dies soll während der Entscheidung dafür sorgen, dass neue Autos, entsprechend des aktuellen Verkehrs, erscheinen.

Die *TrafficLightPhaseCase* weist die Phasenkontrolleinheit mit einer *ApplyTime* von 1.000 Simulationsschritten an, die spezifizierte Phase zu aktivieren. Danach werden weitere 9.000 Simulationsschritte berechnet und dann die Simulation evaluiert. Dazu wird ein *SpeedWaitEvaluator* verwendet:

```
1 public double evaluate(TrafficSimulator simulator) {
2     double result = 0;
3     for (Car car: simulator.getCars()){
4         result += car.getV()*100 - car.getTotalWaitingTime();
5     }
6     return result;
7 }
```

Listing 5.7: Beispiel einer Bewertungsfunktion für die Ampelsteuerung

Entsprechend der Spezifikation des TSDF wird anschließend für den Fall mit den meisten Punkten ein entsprechendes *TrafficLightControlEvent* versendet, der alle Ampeln umstellt.

5.3 Kundenleitsystem

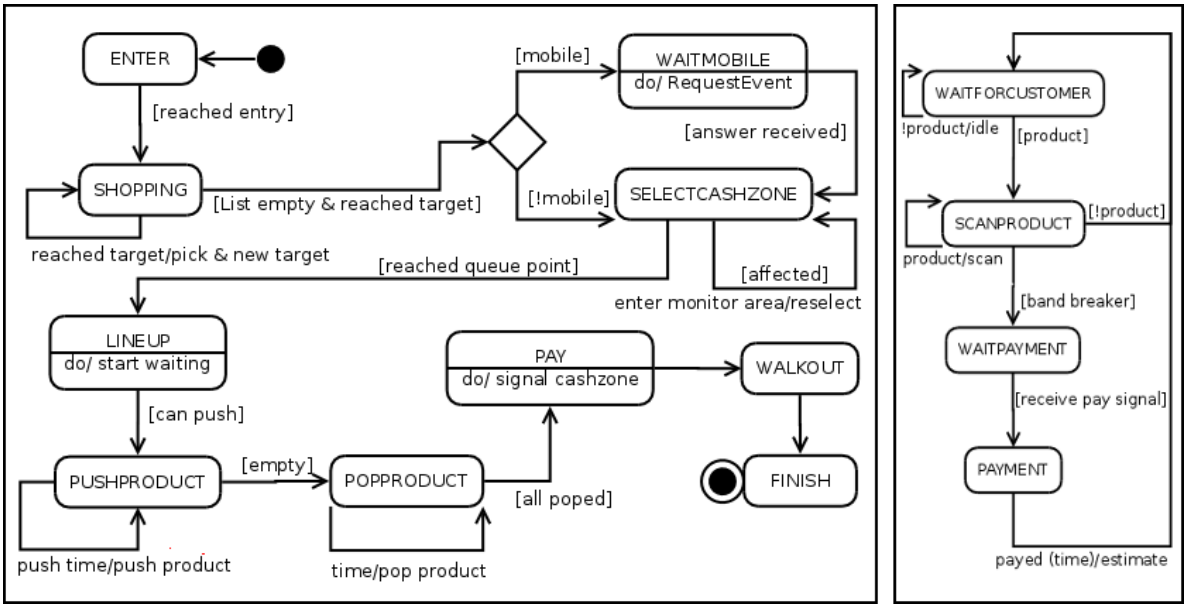


Abbildung 5.4: Automaten zur Simulation eines Kunden (links) und der Automat zur Simulation einer Kasse (rechts)

Die Implementierung des „electronical Cashzone Advisor ServicE“ (kurz: eCASE) reduziert sich auf die Implementierung zweier Automaten, die zur Simulation miteinander kommunizieren. Der Kunde und die Kasse (vgl. Abbildung 5.4). Eine Kasse verfolgt dabei ein einfaches endloses Konzept:

1. **WAITINGFORCUSTOMER**
   In diesem Zustand macht die Kasse nichts und wartet darauf, dass ein Produkt auf das Kassenband gelegt wird. Wird ein Produkt auf dem Band erkannt, wechselt die Kasse in den **SCANPRODUCT** Zustand.

2. **SCANPRODUCT**
   Solange Produkte auf dem Band liegen, bleibt die Kasse in diesem Zustand und befördert die Produkte auf das zweite Band, auf dem der Kunde sie wieder einladen darf. Dabei muss die Kasse je nach Produkt eine gewisse Zeit warten. Wie lange ist im jeweiligen Produkt spezifiziert. Sind keine Produkte mehr auf dem ersten Band, so wechselt die Kasse zurück in den **WAITINGFORCUSTOMER** Zustand. Findet die Kasse einen Band-Trenner, so wechselt die Kasse in den **WAITPAYMENT** Zustand.

3. **WAITPAYMENT**
   Dieser Zustand wartet auf ein Bezahlungssignal vom Kunden. Dies kann eine Weile dauern, da der Kunde zuerst alle Produkte vom zweiten Band nimmt, bevor er dieses Signal sendet. Der Kunde lädt den Band-Trenner ebenfalls ein und erkennt daran, dass er nun bezahlen muss. Da die Kasse diesen zuerst auf das zweite Band geschoben hat, ist sichergestellt, dass die Kasse im richtigen Zustand ist.

4. **PAYMENT**
   Je nach gewählter Bezahlmethode, wird eine gewisse Zeit gewartet, um den Bezahlvorgang zu simulieren. Anschließend springt die Kasse in den **WAITFORCUSTOMER** Zustand zurück.

Dieser stetige Ablauf kann leicht mit weiteren Zuständen erweitert werden, wenn man mehr Details des Kassenvorgangs simulieren möchte. Der Kunde folgt einem komplexeren Automat:

1. **ENTER**
   Zum Anfang befindet sich jeder Kunde in diesem Zustand. Dies bedeutet, dass er alle Hindernisse ignoriert und auf den nächsten Eingang des Ladens zusteuert. Hat er den Eingang erreicht, so wechselt der Kunde in den **SHOPPING** Zustand.

2. **SHOPPING**
   Während dieses Zustands arbeitet ein Kunde seine Einkaufsliste ab. Solange diese nicht leer ist (streichen bedeutet entfernen von der Liste), läuft er zum Produkt und lädt die gewünschte Anzahl ein. Dazu wird eine A^* Wegsuche auf der Basis von Polygonen verwendet. Ist die Einkaufsliste abgearbeitet und alle Produkte eingeladen, so gibt es die Möglichkeit, dass er sein Mobiltelefon nutzt, um eine Kasse zu wählen und deshalb in den **WAITMOBILE** Zustand wechselt. Alternativ wechselt der Kunde in den Kassenwahlzustand **SELECTCASHZONE**. Die Entscheidung wird beim Erstellen eines Kunden entsprechend einer konfigurierten Wahrscheinlichkeit festgelegt, die später evaluiert wird.

3. **WAITMOBILE**
   Nutzt der Kunde das Mobiltelefon, so wird ein Event an das System gesendet, dass nach einer geeigneten Kasse fragt. Dieses kann zusätzlich, je nach Konfiguration, die Position, die Bezahlungsmethode (Bar oder Kreditkarte) sowie eine Einschätzung der Menge der Waren enthalten (Viel, Mittel, Wenig). Es löst einen Entscheidungsprozess im TSDF aus, dessen Antwort aus einem an den Kunden gerichteten Event besteht, welches eine Kassennummer beinhaltet. Sobald diese Antwort erhalten wurde, wechselt er in den **SELECTCASHZONE** Zustand und wendet sich der empfohlenen Kasse zu.

4. **SELECTCASHZONE**
   In diesem Zustand besitzt der Kunde bereits einen Kassenvorschlag. Dieser kann entweder durch das Mobiltelefon oder die geringste Entfernung festgelegt sein. Läuft der Kunde auf dem Weg zur Kasse durch den Einflussbereich eines Monitors, so kann er mit einer gewissen Wahrscheinlichkeit seine Entscheidung entsprechend der Anzeige des Monitors ändern. Diese Wahrscheinlichkeit wird später evaluiert. Kommt die Empfehlung aus einem Smartphone, so haben die Monitore fast keinen Einfluss mehr. Ein Kunde wechselt höchstens drei Mal seine Entscheidung. Dies kann jedoch auch beliebig konfiguriert werden. Ebenfalls beginnt in diesem Zustand eine Zeitmessung, die später durch einen *Evaluator* ausgewertet werden kann. Erreicht ein Kunde eine Kasse, so kann er sich nochmal zwischen dieser Kasse und allen Kassen in seinem Sichtradius entscheiden. Diese Entscheidung basiert allein auf der Anzahl der anstehenden Kunden. Der Sichtradius ist auf

10m gestellt, so dass nur die benachbarten Kassen in Frage kommen. Entschließt sich der Kunde jedoch nicht für einen Wechsel, so wechselt er in den **LINEUP** Zustand und läuft zum Ende der Warteschlange. Um eine Blockierung des Kassenbereichs bei der geringen Intelligenz der simulierten Kunden zu vermeiden, wird hierzu keine Kollisionsprüfung mit Regalen verwendet.

5. **LINEUP**
   Der Kunde bleibt in diesem Zustand, bis er das erste Band der Kasse erreicht. Zu diesem Zeitpunkt prüft er den Platz auf dem Band und wenn Platz verfügbar ist, wechselt er in den Zustand **PUSHPRODUCT**.

6. **PUSHPRODUCT**
   Solange Waren beim Kunden sind, werden diese auf das Band gelegt. Dabei wird immer gewartet, bis genügend Platz auf dem Band ist. Zusätzlich muss je nach Produkt eine gewisse Zeit gewartet werden, um eine Dauer zu simulieren. Die Zeit und der benötigte Platz sind im Produkt spezifiziert. Wie viel Platz auf dem Band frei ist, kann bei der Kasse angefragt werden. Besitzt ein Kunde keine Produkte mehr, so platziert er einen Band-Trenner auf dem ersten Kassenband und wechselt in den **POPPRODUCT** Zustand. Zusätzlich wird beim Auflegen des ersten Produktes ein weiterer Zeitzähler zur Entscheidungsfindung gestartet, der die Wartezeit an der Kasse widerspiegelt.

7. **POPPRODUCT**
   Ist der Kunde an der Reihe und eines seiner Produkte liegt auf dem zweiten Band, so nimmt er es herunter. Dafür wird die gleiche Wartezeit berechnet, wie für das Auflegen eines Produktes. Die Dauer ist im Produkt spezifiziert und kann variieren. Sind keine Produkte für diesen Kunden auf dem zweiten Band verfügbar, so passiert nichts. Nimmt er einen Band-Trenner vom zweiten Band, so erkennt er, dass alle seine Produkte von der Kasse bearbeitet wurden und wechselt in den **PAY** Zustand.

8. **PAY**
   Hier wird lediglich ein Signal an die Kasse übermittelt, danach wechselt der Kunde in den **WALKOUT** Zustand und verlässt den Laden. Die Wartezeit für den Bezahlvorgang simuliert die Kasse, da diese für eine gewisse Zeit keine Produkte bearbeitet. In diesem Zustand werden die Zeitzähler für alle berechneten Wartezeiten gestoppt.

9. **WALKOUT**
   Der Zustand dient einer schönen grafischen Darstellung und hat sonst keine Funktion. Der Kunde bewegt sich in diesem Zustand 10m von der Kasse weg und wechselt dann in den **FINISH** Zustand.

10. **FINISH**
    Der Kunde bleibt in diesem Zustand, bis er vom Simulator entfernt wird. Kunden werden immer am Ende eines Simulationsschrittes entfernt. Befindet sich das System in einer Entscheidungsphase, so wird der Kunde jedoch nicht entfernt, da die in ihm gespeicherten Wartezeiten für die Entscheidung essentiell sind.

```
1  public void simulate(MarketModel market, Random random, long newtime) {
2      int dead = 0;
3      if (walker == null) {
4          //create walker if not done yet
5          walker = new Walker(market.getEntry().getMass(), v / 1000, x, y);
6      }
7      while (time < newtime) {
8          long timedelta = newtime - time;
9          switch (state) {
10             case ENTER:
11                 //Walk to Entry on direct way (even though
12                             //there might be walls!)
13
14                 if (!walker.reachedTarget()) {
15                     walker.walk(timedelta);
16                     this.x = walker.getX();
17                     this.y = walker.getY();
18                     time += Math.round(timedelta - walker.getRestTime());
19                 }
20
21                 if (walker.reachedTarget()) {
22                     //user has entered
23                     if (market.CUSTOMER_ENTER_SENSOR &&
24                          market.hasEventDispatcher()) {
25                         //sensor active and not in simulation
26                         //simulations do not have a dispatcher!
27
28                         market.getEventDispatcher().send(
29                             new NewCustomerEvent()
30                         );
31                     }
32                     state = State.SHOPPING;
33                 }
34
35                 break;
36             case SHOPPING:
37             //other states
38         }
39     }
40 }
```

Listing 5.8: Implementierung der Simulation im Zustandsmodel eines Kunden

Das Quellcodelisting 5.8 zeigt exemplarisch wie der *ENTER* Zustand eines Kunden implementiert ist. Solange das Ziel nicht erreicht ist, wird der Kunde entsprechend der zur Verfügung stehenden Zeit bewegt. Erreicht er das Ziel, so wird der Eingangssensor ausgelöst. Dies darf natürlich nur in der Simulierten Realität geschehen. Diese ist die einzige, die einen *EventDispatcher* besitzt. Zusätzlich wird der Zustand auf *SHOPPING* umgestellt. Die äußerste while-Schleife ermöglicht es, dass mehrere Zustandsübergange in einem Zeitintervall stattfinden können. Dadurch kann eine Simulation bis zu jeder *newtime* durchgeführt werden.

5.3.1 GIS Model

Um die Simulierte Realität sowie die Virtuelle Realität mit einem Grundmodell über die Regale zu versorgen, wird eine GIS-Darstellung (Geoinformationssystem) verwendet. Die Modellierung von Globus-Güdingen, sowie die des IRLs liegen in diesem Format vor und können als Grundlage geladen werden. Das GIS Modell verfügt über Informationen für Regale, Wände, Kassen, Säulen und weitere Hindernisse. Diese können in Polygone umgerechnet werden. Das GIS Modell verfügt nicht über eine Zuordnung von Produktkategorien zu den Regalen. Diese müsste man für die Übertragung zur Wirklichkeit aus einer Datenbank importieren. Da für den Globus bereits eine Produktsuche implementiert ist, sind die Produktpositionen bekannt und können für eine Umsetzung in die Wirklichkeit leicht abgegriffen werden.

Im GIS Model liegen die Punkte als Fließkommazahlen vor, zur Verwendung der Polygonfunktionen in Java müssen diese jedoch in javainterne Polygone überführt werden. Zur Umrechnung wird eine Skalierung auf Zentimeter herangezogen. Alle Positionen und Geschwindigkeiten, die für Kunden angenommen werden beziehen sich daher auf Zentimeter.

5.3.2 Wegsuche

Die Simulation der Bewegung wird von einem Polygon basierten A^* Algorithmus berechnet. Da die Wegfindung parallel für viele hundert Kunden berechnet werden muss, wird der Graph zur Wegberechnung unabhängig von den dazu gespeicherten Werten erstellt. Um zwischen den Regalen einen geeigneten Weg zu finden, wird ein Netz aus verbundenen Punkten auf einer unendlich großen Fläche platziert. Immer wenn am Wegfindungsalgorithmus ein Polygon registriert wird, so werden alle von diesem Polygon zerschnittenen Verbindungen zwischen Punkten berechnet und abgespeichert. Die A^* Wegsuche nimmt später an, dass eine Verbindung zwischen zwei Punkten existiert, solange sie nicht in der Menge der zerschnittenen Verbindungen auftaucht. Diese Menge ist als *HashSet* implementiert und ermöglicht so eine Prüfung, ob ein Weg zwischen zwei Punkten existiert mit einer Komplexität von $O(\log n)$. Die Abstände zwischen den Punkten, können optional eingestellt werden. Damit die Wegsuche terminiert, sollte ein großes Polygon als Rand um das komplette System gezogen werden, da der A^* sonst die unendlich große Grundfläche durchsuchen würde. Als Heuristik für die Suche wird der direkte Abstand verwendet. Auf der Abbildung 3.7 sieht man dieses Randpolygon in hellgrauer Farbe. Die Kassenpolygone, sowie andere Kunden, werden nicht bei der Wegsuche registriert, so das Kunden durch andere Kunden und Kassen durchlaufen können. Registriert werden ausschließlich die Regale sowie das Randpolygon. Da die Wegsuche im Rahmen dieses Ladens unter normalen Umständen nicht länger als 1-2ms braucht, wird zusätzlich eine Zeitschranke von 100ms zur Wegsuche angegeben. Wird diese überschritten, so nimmt der Algorithmus an, dass es keinen Weg gibt. Der Weg kann anschließend optimiert werden, indem die Wegpunkte auf eine direkte Verbindung hin geprüft werden. Dies ermöglicht es, dass Kunden nicht entlang des Rasters sondern direkte Wege laufen, sofern diese existieren. Da diese Optimierung je nach Weg zwischen 5-10ms dauert und nur einen optischen Gewinn für die Laufwege darstellt, wurde diese für die Simulation deaktiviert.

Der Grund dass für die Wegsuche ein eigener Algorithmus entwickelt wurde ist, dass

es zwar viele fertige A^* Implementierungen gibt, jedoch keine gefunden wurde, die mit Polygonen funktioniert und die angestrebte Laufzeit unter 10ms für eine Wegsuche einhält. Dabei ist anzumerken, dass etwa 95% der CPU-Laufzeit für die Wegsuche verloren gehen. Da das Rahmenwerk mehrere hundert Kunden simulieren soll, ist eine Wegsuche mit höherer Laufzeit nicht annehmbar.

Eine mögliche Alternative wäre, den Weg-Graph der Ladenfläche im Voraus zu berechnen. Dies würde aber Schwierigkeiten bei der Platzierung neuer Regale und Monitore erzeugen, da jedes Mal der Graph neu berechnet werden müsste, was sehr lange dauern kann. Andere Möglichkeiten zur realistischeren Wegfindung können der Arbeit von Gerrit Kahl [16] entnommen werden.

5.3.3 Synchronisation

Zur Synchronisation werden mehrere Events verwendet. Das wichtigste Event zur Synchronisation ist die Schätzung der Kassierer, wie viele Kunden an der Kasse anstehen. Diese Schätzung wird immer am Ende eines Bezahlungsvorgangs als Event verschickt und beinhaltet eine Zahl, die bis zu 5% von der Wirklichkeit abweichen kann. In der Virtuellen Realität, werden Kassen bei denen die Anzahl der anstehenden Kunden von der Schätzung abweicht für weitere Kunden geschlossen. Dies führt dazu, dass sich die Zahl der anstehenden Kunden immer der Simulierten Realität anpasst. Zusätzlich werden Kunden bei einem Event über die Position neu platziert und ihr Zustand angepasst. Da eine Zustandsanpassung während des Bezahlvorgangs heikel ist, werden nur Kunden im Zustand **ENTRY** oder **SHOPPING** modifiziert. Findet sich kein Kunde, so wird ein neuer Kunde erzeugt und entsprechend platziert.

Ein weiteres wichtiges Event ist die Eingangsmessung. Hier geht die Arbeit von einem Sensor mit nahezu 100% Genauigkeit aus und sendet für jeden Kunden der Simulierten Realität, der den Laden betritt (also von **ENTRY** auf **SHOPPING** wechselt) ein Event. Dieses Event wird von der Virtuellen Realität genutzt, um ebenfalls einen Kunden zu erzeugen. Dieser wird direkt im Eingang, für den das Signal gesendet wurde platziert. Da es keine geeigneten Sensoren gibt, um die Einkaufslisten zuverlässig zu erkennen, werden die Einkaufslisten zufällig generiert. Eine Optimierung wäre die im Laden bekannte Verkaufsstatistik als Zufallsverteilung anzunehmen. Dies ist ein Beispiel für eine aus Wissen generierte Information. Diese können weiter verbessert werden, wenn die Verteilungen nach Tageszeit, Wochentag und Datum generiert werden. Dazu könnte man alte Kasseninformationen auswerten. Dies war ursprünglich auch geplant wurde aber vernachlässigt, da keine Informationen zur Verfügung gestellt werden konnten.

Ein weiterer interessanter Sensor wäre die vollständige Produktinstrumentierung mit RFID. Diese würden detaillierte Informationen über die Position des Kunden, sowie die Menge der Produkte in den Einkaufswagen liefern. Da dies jedoch nicht realistisch ist, wurde dieser Sensor in der Arbeit vernachlässigt.

Zur Zeitsynchronisation wurde ein anderes Konzept als bei eTAS implementiert, da keine Hypothetische Realität existiert. Die Simulierte Realität sendet ein Zeitevent, welches von der Virtuellen Realität bestätigt wird. Solange ein Event nicht bestätigt wurde, ist die Simulierte Realität angehalten. Diese Modifikation wurde durchgeführt, da bei der Anfrage über Mobiltelefone die Virtuelle Realität sehr viel mehr Laufzeit benötigt. Die Simulationen würden sonst im starken Zeitraffer schnell asynchron verlaufen. Die Anfrage eines Kunden wird in der Regel in weniger als $500ms$ beantwortet,

sofern nicht zu viele Kunden gleichzeitig anfragen. Bei einem in Echtzeit laufenden System wäre dies also kein Problem. Bei einem Zeitraffer mit Faktor 100 hätte dies jedoch erhebliche Konsequenzen, da umgerechnet eine Anfrage fast eine Minute benötigt ($500 \cdot 100 = 50.000ms$). Dazu kommt, dass bei einem Zeitraffer wesentlich mehr Anfragen in gleicher Zeitspanne beantwortet werden müssen. Daher wird die Simulierte Realität pausiert, solange die Virtuelle Realität Berechnungen durchführt.

Die Aktuatoren im Laden können zum einen die Mobiltelefone der Kunden oder die Monitore sein. Für beide gibt es Events, die gewisse Kassen spezifizieren. Ein sogenanntes *MonitorEvent* besagt, dass auf Monitor *mid* die Kasse *cid* bewerben werden soll. Möchte man das System an eine echte Umgebung anschließen, so müsste eine entsprechende Grafik oder Simulation angezeigt werden. Die Mobiltelefone bekommen ein *SelectCheckoutEvent*.

Zusätzlich wurde, um Laufzeit zu Sparen, keine *DecisionNode* verwendet. Die Entscheidungen werden direkt von der Virtuellen Realität berechnet. Dies spart die Serialisierung und Übertragung des relativ großen Modells. Zur weiteren Beschleunigung während dem Zeitraffer wird zusätzlich nicht der *EBS* verwendet, sondern eine ähnlich funktionierende Implementierung ohne Netzwerk. Dies ermöglicht die Evaluationen vieler verschiedener Simulationen zur gleichen Zeit, da keine Netzwerkports benötigt werden. Zusätzlich besitzt diese minimale Infrastruktur keine Filter oder andere Zusatzfunktionen wie serverseitige Services, wodurch weitere *Threads* eingespart und mehr Kapazitäten für die Simulationen frei werden. Die Umstellung auf den EBS ist für die Umsetzung in eine Realität einfach gehalten und stellt, solange das System nicht im Zeitraffer läuft, auch keine Probleme dar.

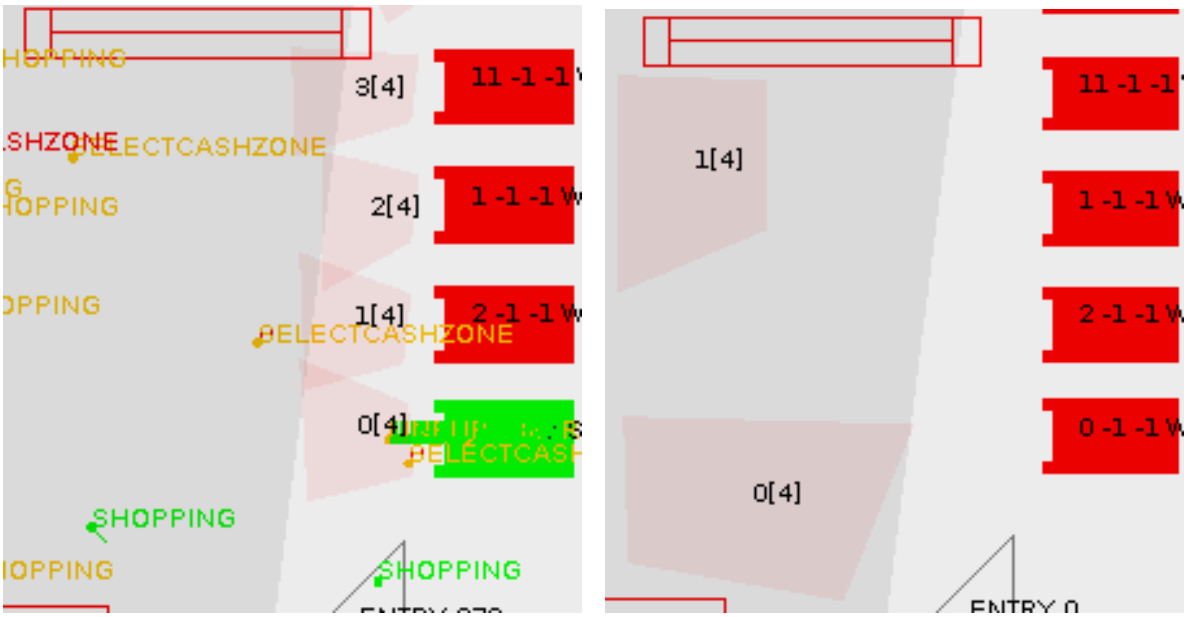
5.3.4 Platzierung der Monitore

Die Platzierung der Monitore erfolgt über die Registrierung eines Polygons, welches den Einflussbereich des Monitors widerspiegelt. So können direkt beim Erzeugen des Modells Einflussbereiche spezifiziert und die Wirklichkeit abgebildet werden. Es wurden zwei Verteilungen getestet: Einmal die Platzierung der Monitore auf den Hauptwegen der Kunden und zum Anderen die Platzierung der Monitore im Bereich jeder Kasse.

Möchte man die Monitore als Hindernis registrieren, so muss ein weiteres Polygon bei der Wegfindung angemeldet werden. Monitore können an der Decke befestigt werden und stellen dann kein Hindernis dar. Daher ist die Registrierung bei der Wegfindung von der Registrierung des Einflussbereichs getrennt.

5.3.5 Entscheidungsfindung

Zur Entscheidungsfindung werden zwei verschiedene Evaluatoren benötigt. Wenn eine Anfrage von einem Mobiltelefon bewertet werden soll, so darf lediglich die Wartezeit dieses Kunden berechnet werden. Dafür wurde der *IndividaulWaitEvaluator* implementiert. Dieser gibt als Bewertung die negative Wartezeit zurück. Zusätzlich kann die Wegzeit zur Kasse ebenfalls mitberechnet werden. Praktisch wird eine Kombination der Wegzeit und der Wartezeit gewählt. Dabei hat die negative Wartezeit im Evaluator die Auswirkung, dass diese vom System für den Kunden minimiert wird. Die negative Wegzeit minimiert zusätzlich die durch den Laufweg von der aktuellen Position zur Kasse benötigte Zeit. Dadurch werden nahe Kassen bevorzugt. Es wird eine Simulation



(a) Monitore für jede Kasse (b) Monitore an den Hauptwegen

Abbildung 5.5: Verteilung der Monitore auf der Ladenfläche (schwach rot eingefärbtes Polygon). Die erste Zahl stellt die Nummer des Monitors dar, die zweite Zahl in eckigen Klammern, die Nummer der beworbenen Kasse.

für jede offene Kasse als Alternative berechnet. Diese Simulation wird auf zehn Minuten festgelegt und die Wahl der Kasse wird direkt angewendet, so dass die vollen zehn Minuten nach der Wahl der Kasse simuliert werden.

Für die Auswahl der Anzeige auf dem Monitor wird ein genetischer Algorithmus mit der Entscheidungsfindung kombiniert. Dazu wird der Simulator kopiert und so eingestellt, dass fertige Kunden nicht mehr aus der Simulation entfernt werden. Anschließend wird die aktuelle Eintrittsrate erfasst und im Simulator festgesetzt. Es werden nun mehrere Monitorkonfigurationen als unterschiedliche Alternativen spezifiziert. Eine Alternative besteht also aus einer Zuordnung aller Monitore zu einer Kasse, die sie bewerben sollen. Damit nicht bei jeder Entscheidung ein Wechsel entsteht, wird die aktuelle Monitorkunfiguration als erste Alternative angegeben. Es muss die erste Alternative sein, da das Entscheidungssystem sich bei gleicher Punktzahl immer für die erste Alternative entscheidet. So wird ein Wechsel bei gleicher Punktzahl vermieden. Zusätzlich werden Mutationen hinzugefügt. Eine Mutation besteht aus der aktuellen Konfiguration, bei der mit einer 70% Wahrscheinlichkeit ein Monitor zufällig verändert wird. Zusätzlich werden drei weitere komplett zufällige Monitorkonfigurationen als Alternativen angeboten. Das macht insgesamt fünf Alternativen zwischen denen das System, unabhängig von der Anzahl der Monitore, entscheiden muss. Für jede Alternative werden zwei Simulationen mit jeweils zehn Minuten simuliert. Die beste Konfiguration wird als Event an die Monitore übertragen und so wird die Konfiguration aktualisiert.

Die Anzahl der Simulationen und der Monitorkonfigurationen wurde so gewählt, dass ein Zeitraffer um Faktor 100 effizient berechnet werden kann. Die Simulationszeit von zehn Minuten wurde aus der üblichen Wartezeit eines Kunden an der Kasse abgeleitet und durch Tests verifiziert. Dies stellt sicher, dass Kunden zwischen Start und Ende der

Simulation auch die Kassenwartezeit überbrücken können. Da bei den Mobiltelefonen nur die Wartezeit des einzelnen Kunden berücksichtigt wird, kann diese im schlimmsten Fall zu -600000 (negative 10 Minuten in Millisekunden) Punkten führen, wenn der Kunde zum Ende der Simulation noch in der Warteschlange steht. Bei den Monitoren dagegen, wird die Wartezeit aller Kunden betrachtet und optimiert.

5.4 Filialleiterunterstützung bei der Kassenöffnung

Die Implementierung von eCOS ist ein optionaler Bestandteil dieser Masterarbeit. Das wesentliche Ziel ist nicht nur die Entwicklung eines Services zur Filialleiterberatung beim Öffnen der Kassen, sondern auch ein Test, wie schnell ein neuer Entscheidungstyp in ein existierendes Modell integriert werden kann.

Zur Entwicklung wurde das bestehende eCASE System kopiert und anschließend die notwendigen Änderungen integriert. Diese sind:

- Implementierung eines Events, welches den Zustand einer Kasse verändert. Normalerweise würde dieses Event den Filialleiter zum Beispiel über eine SMS informieren, dass eine neue Kasse geöffnet oder geschlossen werden soll. Da in unserem Fall jedoch nur eine Simulierte Realität vorliegt, führt das versendete Event direkt zu einer Änderung des Zustands der Kasse. Virtuelle Realität und Simulierte Realität ändern den Zustand der im Event spezifizierten Kasse, sobald ein solches Event empfangen wurde.

- Implementierung eines neuen Evaluators. Wie im Konzept vorgeschlagen wurde ein neuer Evaluator eingeführt, der die Abweichung der Wartezeit zu einer gegebenen Wunschwartezeit evaluiert. Sind keine Kassen geöffnet, so wird $-MAX(long)$ zurückgegeben, was einer sehr großen negativen Zahl entspricht. So ist sichergestellt, dass immer mindestens eine Kasse geöffnet bleibt. Ist mindestens eine Kasse geöffnet, so wird die durchschnittliche Wartezeit aller wartenden Kunden ermittelt. Wenn kein Kunde wartet, ist diese 0 und daher wird dann die negative Anzahl der offenen Kassen zurückgeben. Dadurch wird, sofern keine Kunden warten, die Anzahl der Kassen schrittweise bis zu einer offenen Kasse reduziert. Ansonsten wird die negative Abweichung zum eingestellten Wunschwert zurückgegeben. Dadurch versucht das System die quadratische Abweichung zum Wunschwert minimal zu halten.

- Eine Implementierung der Klasse *Case* wurde hinzugefügt. Diese stellt die Änderung einer Kasse a nach einer Anwendungszeit $x = 5$ Minuten dar. Zur Auflösung dieser Klasse wird das neu eingeführte Event zurückgegeben.

- Die Virtuelle Realität wurde angepasst, so dass sie in regelmäßigen Abständen (10 Minuten) eine neue Entscheidung startet. Dazu wird der aktuelle Zustand, sowie für jede Kasse eine Umkehrung ihres Zustands als Alternative erzeugt. Diese Alternativen werden an das TSDF weitergeleitet und mit jeweils 8 Simulationen je 15 Minuten berechnet.

Die geringe Anzahl an Änderungen zeigt, wie wenig Aufwand die Implementierung einer weiteren Entscheidung bei einem vorhandenen Simulator benötigt. Als Wunschzeit für wartende Kunden wurden 3 Minuten spezifiziert.

6 Evaluation

Im folgenden sollen die Ergebnisse der einzelnen Systeme aufgeführt und erläutert werden.

6.1 Ampelsteuerung

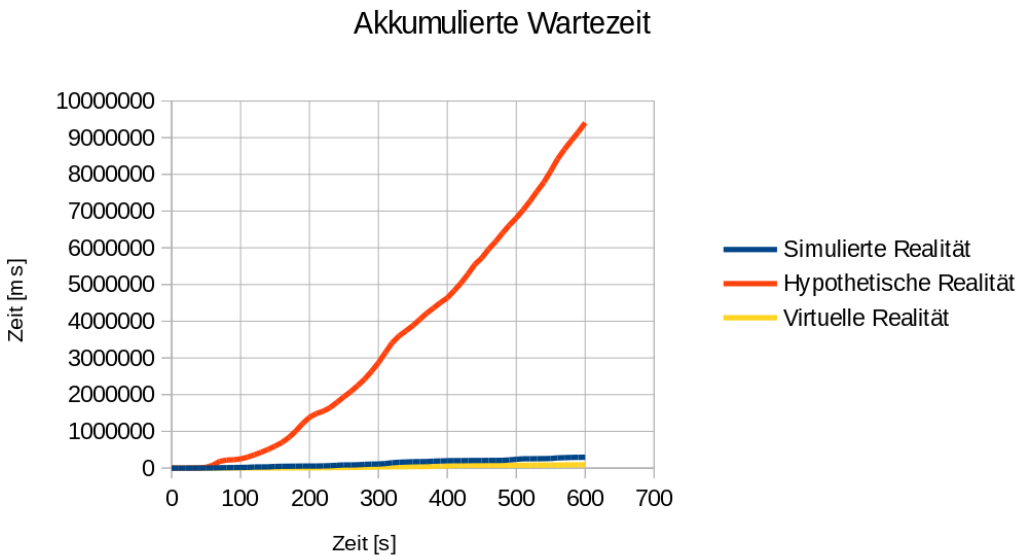


Abbildung 6.1: Ergebnisse der Ampelsteuerung

Grafik 6.1 zeigt die Entwicklung der akkumulierten Wartezeit der Autos in der Simulierten Realität, der Hypothetischen Realität und der Virtuellen Realität. Die Simulierte Realität, stellt dabei die intelligent gesteuerte Kreuzung dar. Die Hypothetische Realität wird durch eine Zeitschaltung gesteuert, erhält die gleichen Sensordaten wie die Virtuelle Realität und erzeugt deshalb die selben Autos. Die Virtuelle Realität stellt das Modell dar, welches durch Sensordaten gewonnen wird und als Basis für die Entscheidung dient. Der Vergleich zwischen Hypothetischer Realität und Simulierter Realität spiegelt den direkten Vergleich der selben Ampel mit intelligenter Steuerung und Zeitsteuerung wider.

Die akkumulierte Wartezeit ist die aufsummierte Wartezeit aller Autos. Dadurch muss die akkumulierte Wartezeit monoton wachsend sein. Der Ursprung liegt bei 0, da zu Beginn keine Autos auf der Kreuzung vorhanden sind. Die Messung wurde mehrmals mit ähnlichen Ergebnissen wiederholt. Im folgenden werden die aus dieser Messung resultierenden Ergebnisse zusammengefasst:

- Die Verkehrslage wird durch das intelligente System besser gesteuert als durch die Zeitsteuerung. Es müssen weniger Autos warten, was für die Umwelt und die Autofahrer positiv ist. Das System fällt also die erwartet guten Entscheidungen.
- Die wachsende Steigung der akkumulierten Wartezeit in der Hypothetischen Realität zeugt davon, dass es einen Stau gibt und immer mehr Autos warten müssen. Dies bezeugt auch die Abbildung 8.2.
- Es gibt eine Abweichung der akkumulierten Wartezeit in der Virtuellen Realität und der Simulierten Realität. Vermutlich kommt es daher, dass die Entscheidungen anhand der Virtuellen Realität gewonnen werden und daher für diese optimiert sind. Das System hat jedoch keine Kenntnis darüber, wie sich die Autos in der Realität entscheiden werden. Ein Linksabbieger blockiert die Kreuzung mehr als ein Rechtsabbieger oder einer der geradeaus fährt. Dies sind Informationen die bei der Entscheidung zwar simuliert werden, jedoch nur anhand von Zufall oder Statistik ermittelt werden können.

6.2 Kundenleitsystem

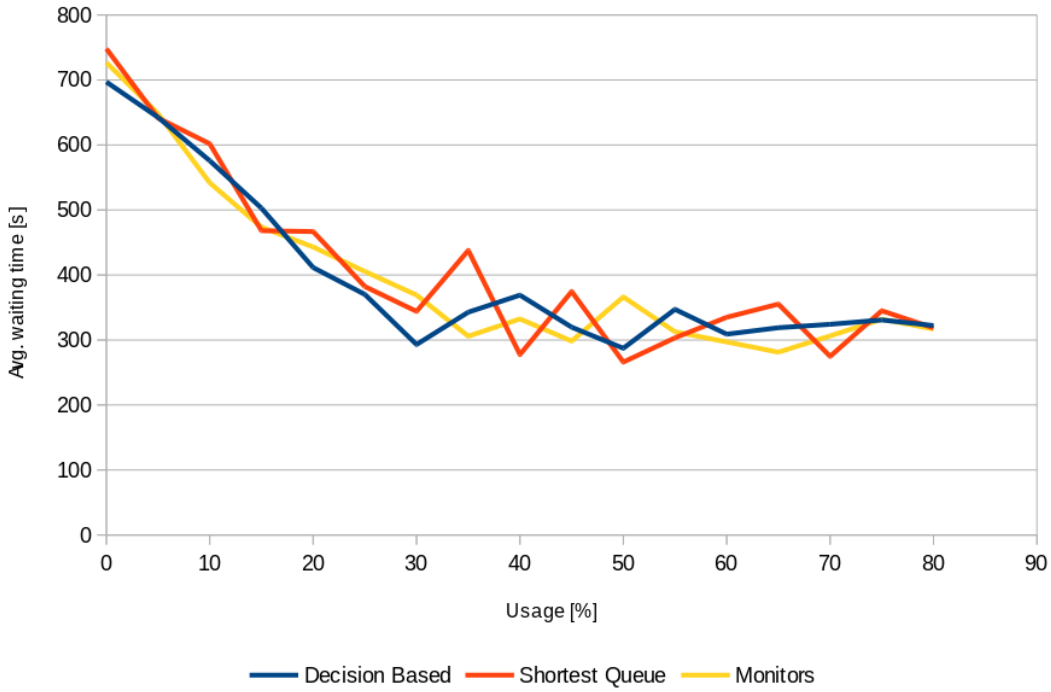


Abbildung 6.2: Vergleich verschiedener Entscheidungsvarianten im eCASE Szenario

Grafik 6.2 zeigt die Parameterauswertung verschiedener Kassensysteme im Vergleich. Die blaue Linie spiegelt dabei die beschriebene App für Smartphones wieder, die eine Anfrage an das intelligente System stellt und entsprechend eine Kassenempfehlung herausgibt. Abgebildet wird die prozentuale Verwendung dieser Anwendung auf die durchschnittliche Wartezeit der Kunden im Laden. Die rote Linie entspricht der blauen Linie, nur wird hier keine intelligente Entscheidung generiert, sondern basierend

auf einer Kassiererschätzung die Kasse mit der geringsten Anzahl wartender Kunden empfohlen. Die gelbe Linie dagegen zeigt die Verwendung von Monitoren an den Hauptwegen und bildet die Einflussrate auf die Entwicklung der durchschnittlichen Wartezeit ab. Die Einflussrate beinhaltet dabei die Wahrnehmung der Monitore, das Vertrauen auf die vorgestellte Entscheidungshilfe und die Verwendung durch die Kunden zusammengefasst als ein prozentualer Wert.

Die Messung wurde in 5% Schritten durchgeführt. Zu jeder Messung wurde der Durchschnitt aus 20 Simulationen von jeweils 4 Stunden simulierter Realzeit verwendet (Dies entspricht $1280 = 20 \cdot 4 \cdot (90/5)$ Stunden Realzeit). Dafür wurde eine gesamte Laufzeit von etwa 132 Stunden Rechenzeit benötigt. Dies entspricht etwa einem Zeitraffer mit Faktor 9.7.

Die folgenden Ergebnisse können aus dieser Messung gewonnen werden:

- Die intelligente Entscheidungsfindung ist nicht messbar besser, als die Empfehlung der Kasse mit der kürzesten Warteschlange. Dies wird später noch genauer untersucht, da es einen Typ von Fällen gibt, in dem die Entscheidung besser sein müsste.
- Die Verwendung einer mobilen App hat etwa den gleichen Einfluss wie die Verwendung von Monitoren.
- Das Kassenverhalten lässt sich durch Entscheidungshilfe optimieren, sofern die Kunden das System verwenden. Sobald etwa 30% das System verwenden, scheint die Optimierung zu konvergieren. Es ist also anzunehmen, dass etwa 300 Sekunden im simulierten Szenario das mögliche Optimum ist. Zumindest aber ist es das Optimum unter der Verwendung der beschriebenen Systeme.
- Bereits bei geringen Nutzungsraten gibt es eine Verbesserung.
- Da die Evaluationen im Zeitraffer ausgewertet wurden, können nicht so viele Simulationen durchgeführt werden, wie in der Realität möglich wären. Daher ist es möglich, dass die Ergebnisse in der Realität besser wären.

Es gibt allerdings einen Fall, bei dem die intelligente Entscheidung besser ist, als die Empfehlung der kürzesten Warteschlange. Dieser spezielle Fall scheint jedoch vernachlässigbar selten einzutreten. Dies tritt immer dann ein, wenn mehrere Kunden gleichzeitig oder in kurzer Abfolge eine Kassenentscheidung über ein Smartphone abfragen. Wogegen die Empfehlung anhand der kürzesten Warteschlange allen Kunden die gleiche Kasse empfehlen wird, so berücksichtigt das intelligente System die Entscheidung der anderen und integriert dies in seine Simulation. Bei der Messung dieser speziellen Situation wurde nicht die durchschnittliche Wartezeit gemessen, sondern die Wartezeit der Kunden. Zusätzlich wurde dafür gesorgt, dass dieser konstruierte Fall in der Simulation bei jedem Durchlauf eintritt, in dem die Einkaufslisten der Kunden festgelegt wurden. Die durchschnittliche Wartezeit aller Kunden, die in dieser Situation eine intelligente Entscheidung bekommen, beträgt 122 Sekunden. Bei der Empfehlung der kürzesten Warteschlange dagegen sind es 184 Sekunden. Dies macht durchschnittlichen Unterschied von 62 Sekunden aus.

Daraus kann gefolgert werden, dass die intelligenten Entscheidungen durchaus besser sind, als die Empfehlung der kürzesten Warteschlange.

6.3 Filialleiterunterstützung bei der Kassenöffnung

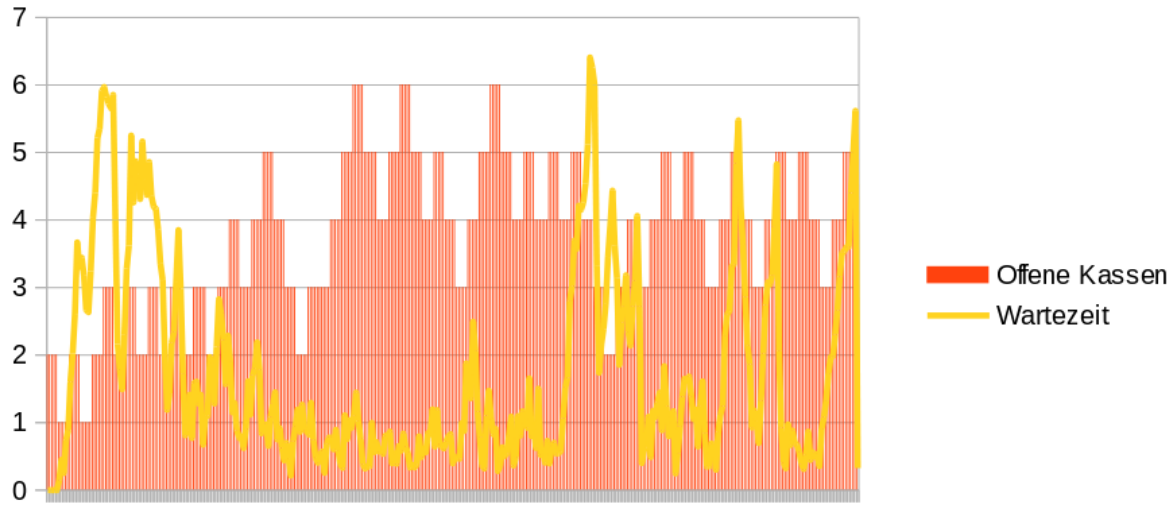


Abbildung 6.3: Entwicklung der offenen Kassen bei eCOS

Abbildung 6.3 zeigt die zeitliche Entwicklung der offenen Kassen sowie die durchschnittliche Wartezeit in Minuten über einen Zeitraum von 12h. Die eingestellte Wunschwartezeit beträgt 3 Minuten. Die durchschnittliche Berechnungsdauer einer Kassenöffnungsentscheidung liegt zwischen 5 und 6 Sekunden. Die durchschnittliche Wartezeit über die kompletten 12h hinweg beträgt 103.64 Sekunden.

Beobachtet man das Verhalten, so stellt man ein interessantes Phänomen fest. Wird die durchschnittliche Wartezeit zu groß, bedeutet dies für den Computer nicht unbedingt, dass eine weitere Kasse geöffnet werden muss. Sammeln sich viele Kunden an einer Kasse und erzeugen so eine hohe durchschnittliche Wartezeit, so schließt der Computer diese Kasse, um weitere Kunden davon abzuhalten sich dort anzustellen.

Insgesamt entstehen bei eCOS jedoch Wartezeiten, die so nicht erwartet wurden. So erreicht die Wartezeit etwa fünf mal über 5 Minuten, einmal über 6 Minuten. Dies hat vermutlich damit zu tun, dass die Entscheidungen nur alle 10 Minuten berechnet werden und in 10 Minuten eine nicht vorhersagbare Entwicklung entstehen kann. Beobachtet man die Entwicklung in der graphischen Benutzeroberfläche, so stellt man fest, dass sich die Wartezeiten in kürzester Zeit erhöhen können. Die hohe Anzahl an offenen Kassen in der Mitte, die trotz geringer Wartezeit existieren, entstehen als Folge des Anstiegs der Wartezeit. So sieht man, dass trotz der Öffnung einer weiteren Kasse ein weiterer Anstieg der Wartezeit beobachtet wird, was bei der nächsten Entscheidung zum Öffnen einer weiteren Kasse führt. Trotz der teilweise nicht erwartet schlechten Wartezeiten, zeigt der Graph eine eindeutige logisch erklärbare Korrelation zwischen den offenen Kassen und der Wartezeit.

Zu bedenken ist ebenfalls, dass beim gewählten Konzept maximal alle 10 Minuten eine Änderung durchgeführt werden kann. Daher ist das System relativ träge, da für das Öffnen von zwei Kassen bereits mehr als 20 Minuten benötigt werden. Dass diese Zeit zu hoch ist, kann man daran feststellen, dass tatsächlich fast alle 10 Minuten auch wirklich eine Änderung durchgeführt wird. In der Abbildung stellt ein roter Strich 2 Minuten dar. Fünf rote Striche bilden eine Entscheidungsperiode. Man sieht also etwa

alle 5 rote Striche eine Änderung der Kassenanzahl. Der durchschnittliche Wert von 103.64 Sekunden Wartzeit ist trotzdem ein gutes Ergebnis.

6.4 Konstante Entscheidungszeit

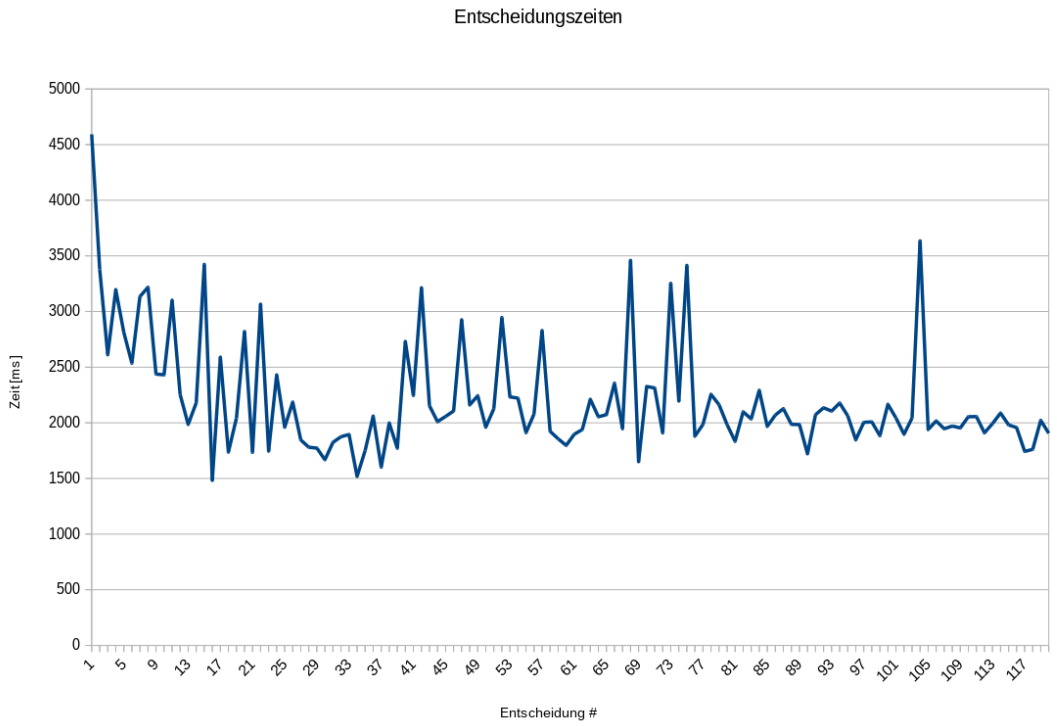


Abbildung 6.4: Zeit, die von den Entscheidungen während der Ampelsteuerung benötigt wurde

Grafik 6.4 zeigt die Laufzeit, welche die verwendete *DecisionNode* zur Berechnung der Ampelphasen benötigt hat. Insgesamt wurden 120 Entscheidungen in den erfassten 10 Minuten der Ampelsteuerung gefällt. Dies entspricht der Einstellung, alle 5 Sekunden eine Ampelsteuerungsentscheidung zu fällen. Die Laufzeit einer Entscheidung variiert zwischen $1,5$ Sekunden und $3,5$ Sekunden, sofern man den ersten Wert von etwa 4.5 Sekunden vernachlässigt, der durch den Start und die beginnende Autooptimierung von Java verursacht wird. Der Mittelwert beträgt 2208 Millisekunden pro Entscheidung. Die Spitzen in der Kurve können auf andere Systemaktivitäten des verwendeten Computers hindeuten. Zur Messung wurde ein mit Linux betriebener Laptop mit einer ein Intel(R) Core(TM) i7 CPU Q 820 @ 1.73GHz mit 16GB Arbeitsspeicher verwendet.

Als Ergebnis kann zusammen gefasst werden, dass die Zeit der Entscheidungsfindung weitestgehend konstant ist, jedoch entsprechend der Situationen schwanken kann.

Da die Anzahl der Autos einen quadratischen Einfluss auf die Laufzeit der Simulation hat, ist damit zu rechnen, dass die Qualität der Entscheidungen in extremen Situationen mit sehr vielen Autos abnehmen wird, um die konstante Laufzeit zu halten. Dies kann durch Tests nicht bestätigt werden: Durch eine Erhöhung der Autofrequenz auf maximal 60 Autos pro Minute aus allen Richtungen, führte zu einer durchschnittlichen

Entscheidungszeit von 2905ms pro Entscheidung und liegt damit noch weit unter den zur Verfügung stehenden 5000ms. Gemessen wurde ebenfalls über einen Zeitraum von 10 Minuten. Zu bedenken gilt, dass die Simulationen visualisiert wurden und das visualisieren vieler Autos in Java ebenfalls ein relevanter Prozess ist, der den Entscheidungen Laufzeit wegnimmt.

Dennoch muss festgehalten werden, dass die Entscheidungszeit maßgeblich von der Situation in der Simulation abhängig sein kann. Um eine Entscheidung in realistischer Zeit zu bekommen, müssen extreme Situationen getestet werden und die Software entsprechend der zur Verfügung stehenden Hardware angepasst werden.

6.5 Auswirkung der Simulationsanzahl

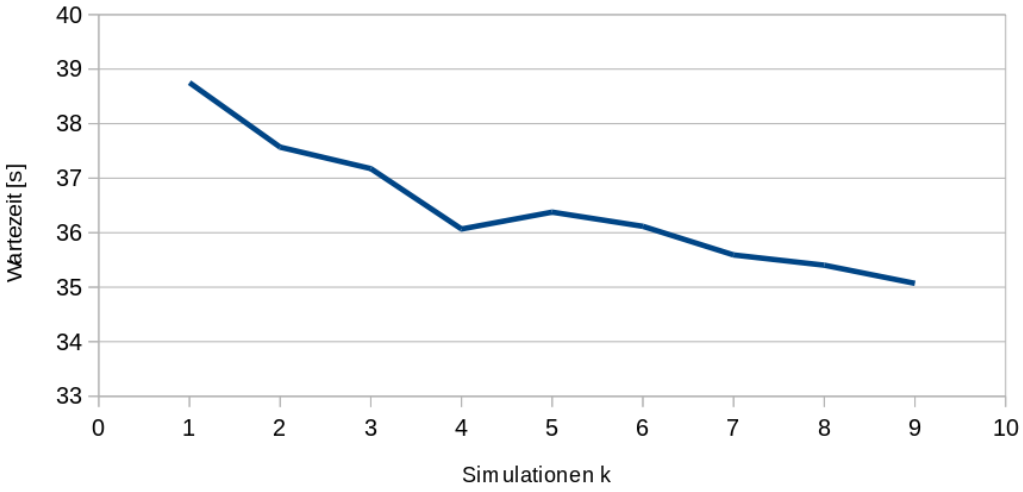


Abbildung 6.5: Zeigt die Entwicklung der Wartezeit an der Kasse durch eine Erhöhung der Simulationen

Abbildung 6.5 zeigt die Entwicklung der Wartezeit eines vereinfachten Kassenszenarios durch Monitorempfehlungen bei einer schrittweisen Erhöhung der Anzahl der Simulationen. Da die Erhöhung der Simulationen sehr zeitaufwendig ist, wurden für jede Simulationszahl nur 4 Messungen mit einer Realzeit von 2 Stunden durchgeführt und der Durchschnitt berechnet. Insgesamt wurden also 72h Realzeit in 12h simuliert. Zur Berechnung wurde ein Server mit einem Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz mit 4 Kernen und Hyperthreading mit 32GB Arbeitsspeicher verwendet.

Die Grafik zeigt eine eindeutige Verringerung der Wartezeit an den Kassen. Durch die größere Anzahl an Simulationen können die Monitore schneller eine für den Kassenbereich optimale Einstellung erzielen und damit die Wartezeit verringern. Dies war nach den mathematischen Berechnungen zu erwarten, dennoch konnten durch die Verwendung von 9 Simulationen im Schnitt nur 3 Sekunden Wartezeit eingespart werden. Dies bedeutet, dass die optimale Wartezeit nahezu erreicht wurde oder wesentlich mehr Simulationen für eine weitere Steigerung benötigt werden. Die Verwendung von 9 Simulationen für jede Alternative erhöht die Laufzeit jedoch drastisch. Daher ist eine derart große Anzahl an Simulationen bei einem Gewinn von 3 Sekunden nicht gerechtfertigt. Dennoch konnte die erwartete Entwicklung gezeigt werden.

Die leichte Erhöhung bei 5 Simulationen sowie die geringe Absenkung bei 6 Simulationen können durch Schwankungen entstanden sein, da jeweils nur 4 Messungen berechnet wurden. So wurde bei den 4 Messungen der 5 Simulationen einmal ein stark abweichender Wert von fast 38 Sekunden gemessen, der den starken Ausbruch erklären könnte. Bei einem Wert von 6 Simulationen gibt es bei der Berechnung des Durchschnitts ebenfalls einen Ausreißer von 37 Sekunden.

6.6 Platzierung der Monitore

Zur Platzierung der Monitore wurden zwei Szenarien getestet. Einmal die Platzierung der Monitore an den Hauptwegen und die Platzierung der Monitore an jeder Kasse. Zur Ermittlung der Hauptwege wurde das Kundenverhalten über einen Zeitraum von 30 Minuten in der Simulierten Realität beobachtet. Die Abbildung 8.4 zeigt die daraus gewonnenen Hauptwege, sowie die Platzierung der Monitore. Die Abbildung 6.6 zeigt die Positionierung der Monitore für jede Kasse im Vergleich zur Darstellung der Monitore an den Hauptwegen.

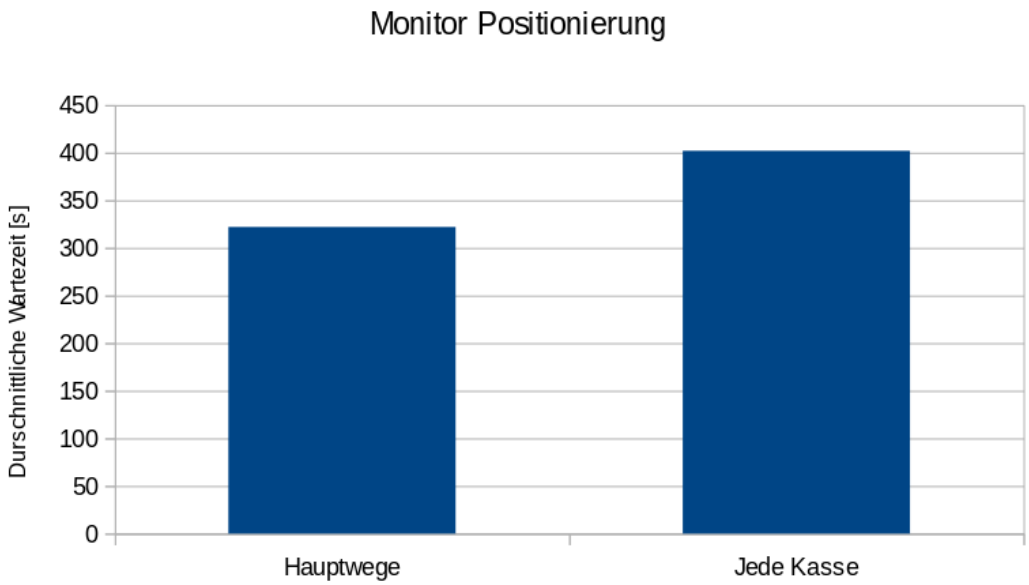


Abbildung 6.6: Vergleich verschiedener Entscheidungsvarianten im eCASE Szenario

Die Abbildung 6.6 zeigt die durchschnittliche Wartezeit, die innerhalb von 4 Stunden bei den unterschiedlichen Monitorpositionen gemessen wurde. Die Auswirkung der Monitore wurde dabei konstant auf 35% gesetzt. Dieser Wert entspricht einem realistischen Szenario und erreichte in der Evaluation einen nahezu optimalen Wartezeitwert. Die folgenden Ergebnisse können aus der Messung gewonnen werden:

- Die Positionierung der Monitore spielt eine Rolle bei deren Wirkung.
  Dies ist ein erwartetes Ergebnis und bedarf daher keiner weiteren Kommentierung.

- Die Platzierung von wenigen Monitoren an den Hauptwegen liefert ein besseres Ergebnis. Vermutlich führt die Wahrnehmung mehrerer Monitore beim Kunden

zu Verwirrung und verleitet ihn zum mehrmaligen Wechsel der Kasse. Dies führt zu einer erhöhten Wartezeit.

- Die Positionierung der Monitore kann durch simulierte Tests gewonnen werden. Dadurch kann unter der Annahme von Realitätsnähe auf das manuelle Ausprobieren von Monitorpositionen verzichtet werden.

7 Konklusion

In dieser Arbeit wurde gezeigt, wie man mit Hilfe von Dual Realty und Trajektoriensimulation intelligente Entscheidungen in Echtzeit durchführen kann. Dazu werden die Sensordaten einer instrumentalisierten Realität verwendet, um ein synchrones virtuelles Model dieser Realität zu erstellen. Diese Virtuelle Realität wird benutzt, um zufällige Simulationen verschiedener Alternativen vollautomatisch durchzuführen. Die durch Simulation gewonnenen Trajektorien werden dann bewertet und benutzt, um die Alternativen in der Realität mit Hilfe von Aktuatoren umzusetzen. Da instrumentalisierte Umgebungen teuer sind, wurde in dieser Arbeit auf sogenannte Simulierte Realitäten zurückgegriffen. Also Simulationen, die eine echte instrumentalisierte Umgebung widerspiegeln sollen.

Diese Methode der Entscheidungsfindung ist jedoch nicht auf Duale Realitäten beschränkt. Sie kann auch zur Erstellung von KIs in Computerspielen oder beim Bau autonomer Roboter Anwendung finden. Generell kann die Methode immer dann verwendet werden, wenn verschiedene Alternativen mit Hilfe eines Modells simuliert werden und die dadurch entstandenen Trajektorien mit Hilfe einer geeigneten Bewertungsfunktion automatisch evaluiert werden können.

Es konnte gezeigt werden, dass die durch Trajektoriensimulation gewonnenen Echtzeitentscheidungen einen positiven Einfluss auf die instrumentalisierten Umgebungen haben. In allen drei Anwendungsfällen wurden positive Ergebnisse festgestellt. Allerdings muss die Trajektoriensimulation nicht immer die geeignetste Methode sein um eine Entscheidung zu fällen. So kann beim eCASE auch die statische Empfehlung der kürzesten Warteschlange zu einer nennenswerten und gleichwertigen Verbesserung der Situation führen, wird jedoch mit wesentlich geringerem Aufwand gewonnen.

Eine große Fehlerquelle kann die abweichende Simulation sein. Da in dieser Arbeit nur Simulierte Realitäten verwendet wurden, die im wesentlichen auf dem gleichen Simulator aufsetzen wie die gewonnenen Entscheidungen, tritt dies in dieser Arbeit jedoch nicht auf. Es ist jedoch denkbar, dass bei der Verwendung von echten instrumentalisierten Umgebungen die Simulationen stärker von der Realität abweichen und daher die Entscheidungsfindung negativ beeinflussen. Allerdings zeigen die Ergebnisse von eCASE, bei die Virtuelle Realität stark von der Simulierten Realität abweicht, dass auch bei größeren Abweichungen noch gute Ergebnisse produziert werden können. Es sollte also immer auf ein geeignetes Level der Abstraktion bei der Implementierung eines Simulators geachtet werden.

7.1 Erreichte Ziele

Zu Beginn wurden fünf wesentliche Ziele für diese Masterarbeit aufgestellt. Im Folgenden soll nun geprüft werden, inwiefern diese Ziele erreicht wurden:

- **Die Entwicklung eines Rahmenwerks**

  Aus der Konzeption sowie der mathematischen Definition konnte ein in Java entwickeltes Rahmenwerk TSDF erstellt werden, welches die vorgestellten Trajektoriensimulationen vereinfacht und die Verwendung und Implementierung anleitet. Dieses Rahmenwerk konnte in einem Testszenario sowie zwei Anwendungsszenarien erfolgreich getestet werden. Die Ausführung wird automatisch parallelisiert, damit Mehrkernprozessoren optimal ausgenutzt werden. Zusätzlich wurde ein EBS *ClientService* entwickelt. Wird dieser auf einem Computer gestartet, so können Entscheidungen an diesen Computer delegiert werden. Dies ermöglicht die Verteilung von Entscheidungen im Netzwerk ohne großen Mehraufwand. Dabei muss allerdings berücksichtigt werden, dass für jede Entscheidung das komplette Modell übertragen werden muss.

- **Die Entwicklung eines Testszenarios**

  Um das Rahmenwerk zu entwickeln wurde das Testszenario eTAS entwickelt. Der eTAS bietet die Möglichkeit eine in der Anwendung erstellte Kreuzung mit Ampeln in Echtzeit intelligent zu steuern. Die Ergebnisse, die bei eTAS gemessen wurden zeugen davon, dass der Ansatz der Trajektoriensimulationen geeignet ist komplexe Entscheidungen in Echtzeit durchzuführen. Der eTAS verwendet Sensoren um Autos auf der Kreuzung sowie deren Geschwindigkeit zu erfassen. Dadurch erzeugt er eine Virtuelle Realität, die etwa mit der Wirklichkeit übereinstimmt. In einem fest gesetzten Intervall führt eTAS Trajektoriensimulationen auf einer ausgelagerten *DecisionNode* Entscheidungen über die nächste einzustellende Ampelphase durch. Die Wartezeiten reduzieren sich dabei drastisch gegenüber einer auf Zeit basierenden Ampelsteuerung. Besonders bei wenig Verkehr kann man beobachten, wie Autos fast ohne zu Bremsen die Kreuzung passieren können, da das System explizit für diese Autos umschaltet. Bei viel Verkehr passt das System sich den gegebenen Situationen automatisch an und kann flexibler als eine Zeitsteuerung die Wartezeiten optimieren.

- **Die Entwicklung eines Kundenleitsystems**

  Das für diese Arbeit ursprünglich angedachte Ziel, ein Kundenleitsystem für Globus GmbH zu konzipieren und einen Prototypen zu implementieren, konnte umgesetzt werden. Es stellt sich jedoch heraus, dass Trajektoriensimulationen praktisch nicht messbar bessere Entscheidungen als ein einfacher statischer Algorithmus fällen können. Empfiehlt ein System immer die Kasse mit der kürzesten Warteschlange, so kann eine ebenso gute Optimierung erzielt werden. Es konnte jedoch ein

Spezialfall konstruiert werden, bei dem mehrere Kunden gleichzeitig eine Kassenauswahl über ihr Mobiltelefon treffen, bei der die Trajektoriensimulation bessere Entscheidungen produziert. Dieser tritt in der Realität jedoch so selten auf, dass der Unterschied nicht messbar ist. Dies spricht jedoch nicht gegen die Trajektoriensimulation, sondern ist viel eher ein Indiz dafür, dass es keine bessere Optimierung mit den gegebenen Informationen gibt. Dass die Trajektoriensimulation das gleiche Ergebnis findet zeigt eher, dass diese in der Lage ist gute Entscheidungen zu treffen. Daher ist die Entwicklung eines Kundenleitsystems dennoch ein Erfolg - auch wenn die Empfehlung für Globus der statische Algorithmus wäre.

- **Die Entwicklung eines Kassenöffnungssystems**

  Die Entwicklung eines Systems eCOS, das Filialleiter beim Öffnen und Schließen von Kassen unterstützt, war ein optionales Ziel. Es konnte vollständig erfüllt werden und ist zusätzlich ein Indiz dafür, wie schnell man weitere Entscheidungsmöglichkeiten auf einer bereits implementierten Datenbasis umsetzen und verwirklichen kann. Die Qualität des Assistenzsystems zeigt einen guten Durchschnitt bei der zu erzielenden Wartezeit, hat jedoch teilweise hohe Schwankungen, die auf die Dauer der Entscheidungsumsetzung zurückgeführt werden können. So können bei einer eingestellten optimalen Wartezeit von 3 Minuten durchaus Schwankungen bis zu 6 Minuten durchschnittlicher Wartezeit gemessen werden.

- **Die mathematische Fundierung.**

  Das Konzept der Trajektoriensimulation konnte vollständig mathematisch definiert werden. Dazu wurde die „k-besser“-Relation eingeführt und erklärt. Dadurch können Alternativen einer Entscheidung mathematisch verglichen werden. Die Trajektoriensimulation stellt dabei theoretisch eine Einschätzung des Erwartungswerts einer Bewertungsfunktion dar. Es konnte definiert werden, wann eine Entscheidung zwischen zwei Alternativen richtig oder falsch ist und es wurde gezeigt, wie man dies verwendet um Vorhersagen durchzuführen. Dadurch, dass die Trajektoriensimulationen eine Approximation des Erwartungswertes durch Testsimulationen darstellt, ist klar, dass mehr Simulationen diesen Erwartungswert besser approximieren. Insgesamt führt also eine Erhöhung der Anzahl der Simulationen zu einem besseren Ergebnis, was auch praktisch gemessen werden konnte. Da Simulationen auf mehrere Kerne eines Prozessors verteilt werden, Computer in der Zukunft nur noch mehr Kerne jedoch keinen Geschwindigkeitszuwachs mehr erzielen werden und die Simulation und Bewertung optimal parallelisierbar ist, ist Trajektoriensimulation eine zukunftssichere Entscheidungsmethode.

7.2 Weiterführende Arbeiten

Die Entwicklung des TSDF muss als erste Implementierung eines neuen KI-Systems zur Entscheidungsfindung interpretiert werden. Im Rahmen einer Masterarbeit können nicht alle Details einer neuen Idee analysiert und getestet werden. So fehlt zum Beispiel der Test des Rahmenwerks in einer echten instrumentalisierten Umgebung. Dafür bieten sich mehrere instrumentalisierte Umgebungen an, wie zum Beispiel eine Ampel. Da das Rahmenwerk nur die Phasen steuert, erhöhen sich die Unfälle bei Fehlern nur geringfügig. Es ist nicht möglich, dass gefährliche Ampeln zur gleichen Zeit grün sind, sofern sie nicht in einer Phase zusammengeschlossen sind. Die leicht erhöhte Gefährdung kommt durch die unbekannte Schaltlogik für die Autofahrer, die normal zeitgesteuerte Ampeln gewohnt sind und zum Beispiel nicht damit rechnen, dass die Ampel unterschiedlich schnell in andere Phasen wechselt. Eine Implementierung, die das quelloffene Verkehrsentscheidungsrahmenwerk SUMO verwendet, würde weitere Indizien zur Vergleichbarkeit mit anderen Algorithmen zur Ampelsteuerung liefern und wäre deshalb ein wünschenswertes Ziel. Ein weiteres Beispiel für eine instrumentalisierte Umgebung wäre ein großer Supermarkt, um das Kassenleitsystem zu testen. Allerdings liegt die Vermutung nahe, dass eine nützliche Instrumentalisierung für einen großen Supermarkt zu teuer ist und lieber auf ein Empfehlungssystem für die kürzesten Warteschlangen zurückgegriffen wird.

Da die verwandten Monte-Carlo-Simulations-Bäume sehr erfolgreich für die Entwicklung von Computerspielen eingesetzt werden, wäre ein direkter Vergleich und Verwendung der Trajektoriensimulationen in Computerspielen ein vielversprechender Ansatz. Die hier vorgestellten Trajektoriensimulationen sollten vor allem für Computerspiele, die nicht rundenbasiert sind, geeignet sein. Ferner können durch die implementierte Fehlerbehaftete k-besser Relationen leicht Computergegner mit unterschiedlicher Stärke implementiert werden, die auch falsche Entscheidungen treffen.

Ein weiteres Einsatzfeld wäre die Robotik. Autonome Roboter können über Sensoren virtuelle Realitäten erzeugen, die zur Entscheidungsfindung über Trajektoriensimulation genutzt werden könnten. Der Roboter könnte so Entscheidungen durchdenken und rationale Entscheidungen treffen.

Natürlich ist auch der direkte Vergleich mit anderen KI-Systemen in gleichen Szenarien eine interessante Zielrichtung. Dazu müsste die gleiche Entscheidungsproblematik in verschiedenen Systemen implementiert werden und dann Qualität, Geschwindigkeit und vor allem die Qualitätssteigerung durch Parallelisierung verglichen werden. Zur weiteren Qualitätssteigerung durch Parallelisierung wäre es empfehlenswert, das Rahmenwerk so zu erweitern, dass eine einzige Entscheidung effizient auf viele Computer verteilt werden kann. Die parallele Ausführung von Simulationen auf mehreren Computern birgt wenig Schwierigkeiten bei der Synchronisation. Interessant wäre der Vergleich jedem Computer eine eigene Virtuelle Realität zu geben, mit der Möglichkeit, die Virtuelle Realität zusammen mit dem Simulationsauftrag zu übertragen und Ergebnisse zentral auszuwerten. Die erste Möglichkeit hätte den Vorteil, dass unterschiedliche Virtuelle Realitäten als Basis der Entscheidung dienen, was Fehler durch die Sensorauswertung minimiert, da mehr Variationen vorhanden sind und dies die Realität besser approximieren könnte. Die zweite Möglichkeit würde lediglich die Geschwindigkeit für Simulationen erhöhen und so die Anzahl der in einem Zeitraum durchführbaren Simulationen linear mit der Anzahl der zur Verfügung stehenden Computer ansteigen lassen. Wie bewiesen würde

dies zu einer Qualitätssteigerung führen.

Die mathematische Theorie hinter Trajektoriensimulationen steckt noch in den Kinderschuhen. Es stellt sich die Frage, ob es möglich ist, genauere Aussagen über die Auswirkung der Anzahl der Simulationen zu beweisen und nicht nur festzustellen. Da sich dies im wesentlichen auf Schätzungen des Erwartungswertes reduziert, ist anzunehmen, dass mit bereits fundierten und erwiesenen Theorien der Stochastik weitere Eigenschaften der Trajektoriensimulation abgeleitet werden können.

Danksagung

Mit diesen Zeilen möchte ich nun noch allen danken, die mir bei dieser Arbeit zur Seite gestanden haben. An erster Stelle möchte ich hier meinen Betreuer Gerrit Kahl nennen, der mich stets forderte und mir mit seinem Wissen zur Seite stand. Außerdem möchte ich auch meinen Eltern Dr. Hans-Jürgen Bürckert und Silvia Bürckert danken, die mein Studium finanzieren. Ansonsten möchte ich noch Marius Narkus, Ulrike Rheinheimer, Christine Pyttlik und Horst Giesemann für die ausführlichen Rechtschreibprüfungen danken und möchte jedem noch fündigen Leser anbieten, die verbleibenden Rechtschreibfehler doch gerne für sich zu behalten.

Natürlich möchte ich auch der DFKI GmbH und der Globus GmbH danken, da sie die für meine Arbeit notwendigen Materialien zur Verfügung gestellt haben und in der Zukunft die Ergebnisse dieser Arbeit möglicherweise umsetzen werden.

Zu guter Letzt danke ich meiner Freundin, meinen Freunden, meinen Mitstudenten, meiner Mitbewohnerin und sonstigen Menschen, die mich über die ganze Zeit der Ausarbeitung und den letzten sechs Monaten bis zur Abgabe hin ertragen mussten.

Mögen sie ihren Frieden wieder finden.

Saarbrücken 8. Juni 2014

Christian Felix Bürckert

8 Appendix

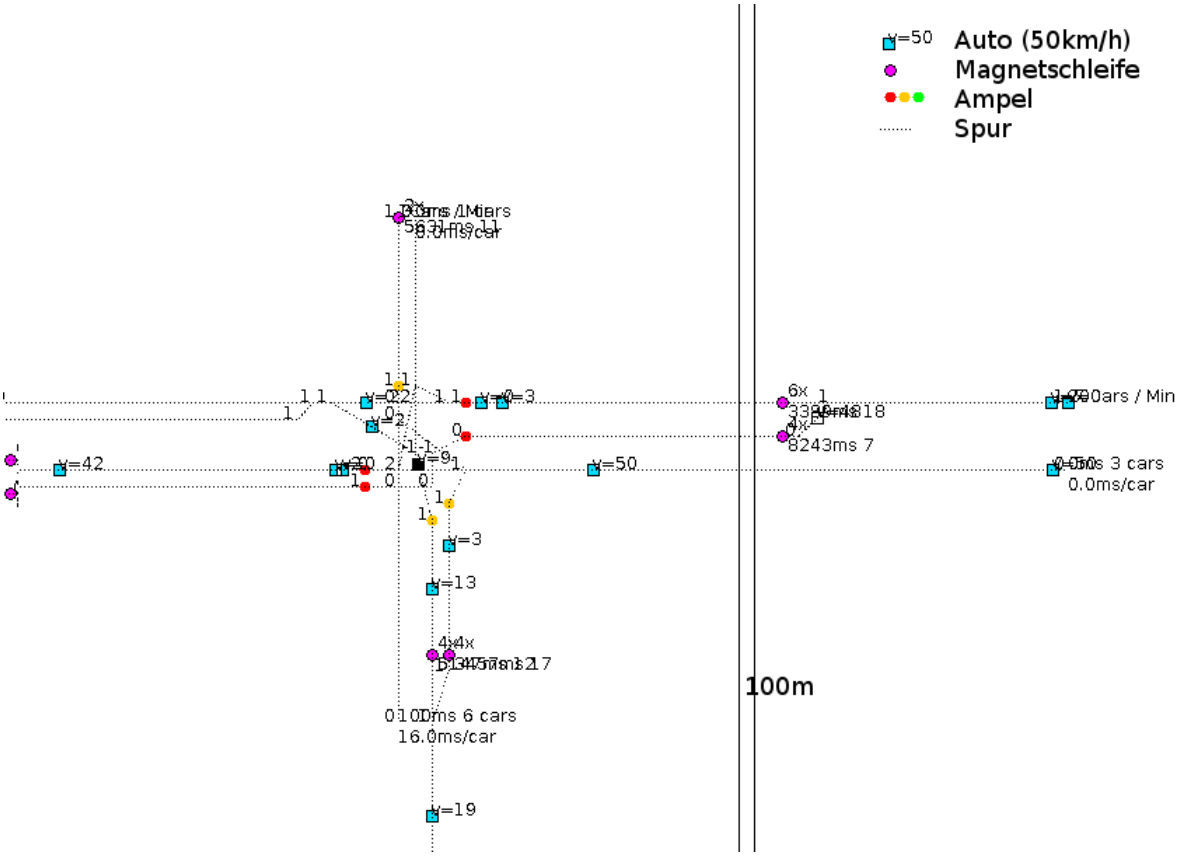


Abbildung 8.1: Modifizierter Screenshot der Graphischen Oberfläche. Dargestellt wird die Kreuzung in der Simulierten Realität bei der Ampelsteuerung.

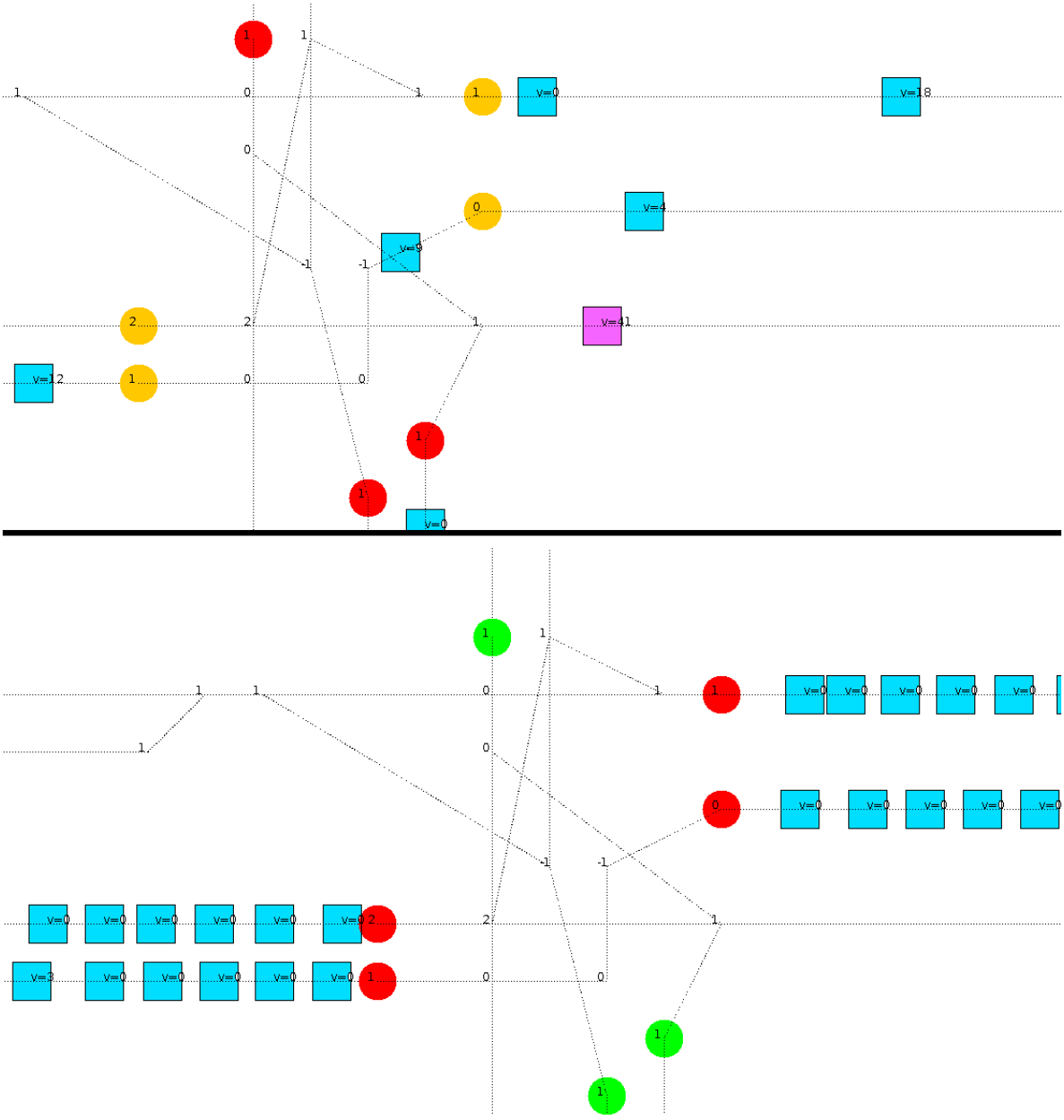


Abbildung 8.2: Verkehrssituation nach 7 Minuten der Messung in der Simulierten Realität (oben) und der Hypothetischen Realität (unten). Dies stellt einen direkten Vergleich der Kreuzung mit und ohne Trajektoriensimulation dar.

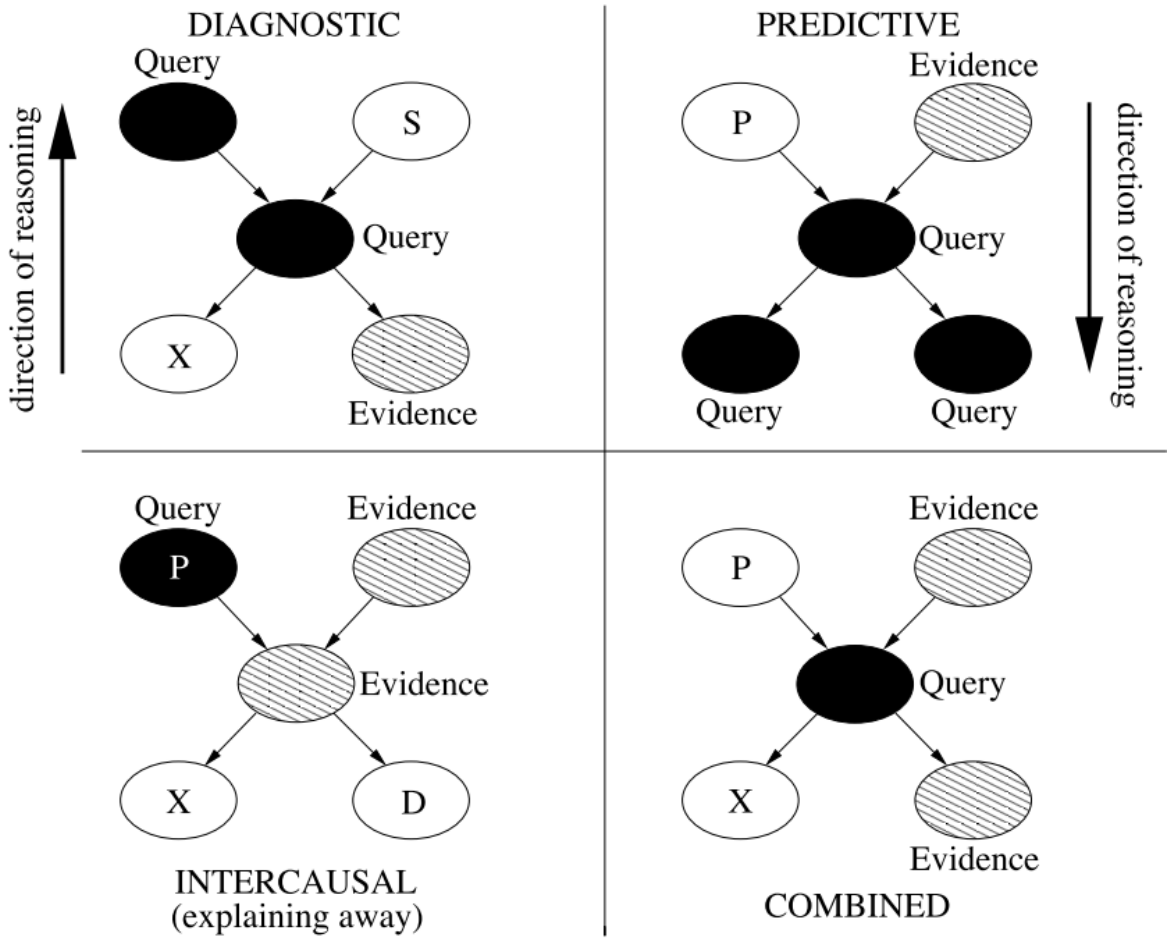


Abbildung 8.3: Begründungsvarianten in einem Bayesschen Netz nach A. Korb, Kevin B and Nicholson[24]

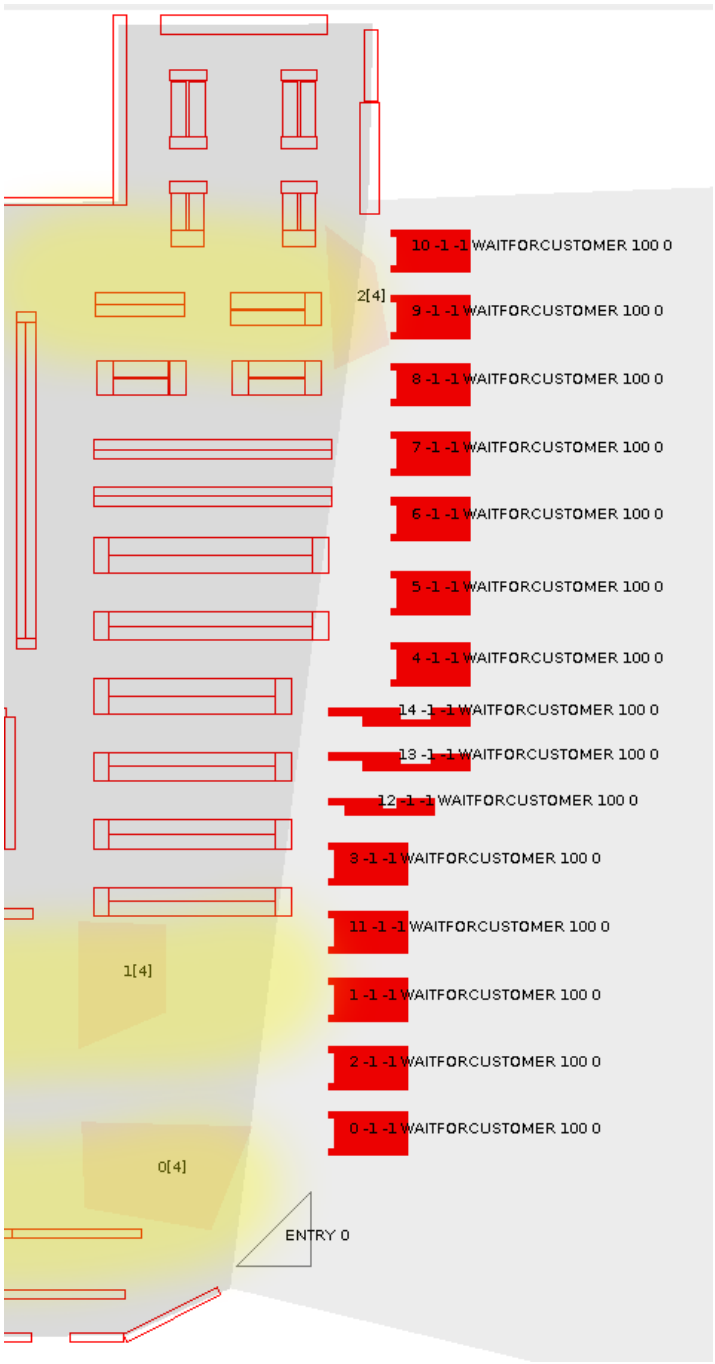


Abbildung 8.4: Darstellung der ermittelten Hauptwege in der Simulierten Realität im Warenhaus Globus Güdingen. Diese müssen nicht mit der Wirklichkeit im Warenhaus Güdingen übereinstimmen.

Literaturverzeichnis

[1] Anurag Agrahari, SK Tripathi, and Dr. ShashiKant Tripathi. A Theoretical Framework for Development of Decision Support System for Agriculture. Research Inventy ISSN: 2278-4721, 1(6):50–55, 2012.

[2] Albert AA Angehrn and Tawfik Jelassi. DSS research and practice in perspective. Decision Support Systems, 12(4–5):267–275, 1994.

[3] Morrin R Forin B and Archer W. Information processing behaviour, the role of irrelevant stimulus information. Journal of Experimental Psychology, 61, 1961.

[4] Joris Borsboom, Jahn-Takeshi Saito, Guillaume Maurice Jean-Bernard Chaslot, and Jos W H M Uiterwijk. A Comparison of Monte-Carlo Methods for Phantom Go. In Proc. BeNeLux Conf. Artif. Intell., pages 57–64, Utrecht, Netherlands, 2007.

[5] J S Bruner, J J Goodnow, and G A Austin. A study of thinking. John Wiley, New York, 1956.

[6] Guillaume Maurice Jean-Bernard Chaslot. Monte-carlo tree search. Phd, ISBN 978-90-8559-099-6, 2010.

[7] Jongeun Choi, Songhwai Oh, and Roberto Horowitz. Distributed learning and cooperative control for multi-agent systems. Automatica, 45:2802–2814, 2009.

[8] Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte Carlo planning in RTS games. CIG, pages 117–124, 2005.

[9] Takeshi Fukase, Yuichi Kobayashi, and Ryuichi Ueda. Real-time decision making under uncertainty of self-localization results. RoboCup 2002, pages 375–383, 2003.

[10] A Gachet. Building Model Driven Decision Support Systems with Dicodess. Informatik-Dissertationen ETH Zürich. Zuerich VDF Hochschulverlag, 2004.

[11] Dirk Gehrig and Hildegard Kuehne. Hmm-based human motion recognition with optical flow data. Humanoid Robots, 2009, pages 425–430, 2009.

[12] Robert P Goldman, Christopher W Geib, Henry Kautz, and Tamim Asfour. Plan Recognition. Technical Report 4, Dagstuhl, 2011.

[13] Robert RL Gordon, Warren Tighe, and ITS Siemens. Traffic control systems handbook. Reproduction, 1(October):367, 2005.

[14] G A Gorry and M S Scott-Morton. A framework for management information systems. Sloan Management Review, 13(1):55–71, 1971.

[15] Haettenschwiler. Neues Anwenderfreundliches Konzept der Entscheidungsunterstützung. Gutes Entscheiden in Wirtschaft, Politik und Gesellschaft. Zuerich VDF Hochschulverlag, pages 189–208, 1999.

[16] Gerrit Kahl. Masterarbeit über PEG und MaMiNa:, 2007.

[17] Gerrit Kahl and Christian Bürckert. Architecture to Enable Dual Reality for Smart Environments. In The 8th International Conference on Intelligent Environments. International Conference on Intelligent Environments (IE-12), The 8th, June 26-29, Guanajuato, Mexico. IEEE, 2012.

[18] Gerrit Kahl, Christian Bürckert, and Spassova Lübomira. Complex Event Processing in Cyber-Physical Systems, 2013.

[19] Gerrit Kahl, Christian Bürckert, Lübomira Spassova, and Tim Schwartz. Event Broadcasting Service – An Event-Based Communication Infrastructure. In Proceedings of the Second International Workshop on Location Awareness for Mixed and Dual Reality (LAMDa'12). Workshop on Location Awareness for Mixed and Dual Reality (LAMDa-12), located at IUI 2012, February 14-17, Lissabon, Portugal, pages 9–12. online, 2012.

[20] Gal A Kaminka, Pedro U Lima, and Raúl Rojas, editors. RoboCup 2002: Robot Soccer World Cup VI, volume 2752 of Lecture Notes in Computer Science. Springer, 2003.

[21] Takeo Kanade, Peter Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. IEEE Multimedia, 4:34–47, 1997.

[22] Tan Kok Khiang, Marzuki Khalid, and Rubiya Yusof. Intelligent traffic lights control by fuzzy logic. Malaysian Journal of Computer Science, 9:29–35, 1996.

[23] G Koole and O Passchier. Optimal control in light traffic Markov decision processes. Mathematical methods of operations research, pages 63–79, 1997.

[24] Ann E Korb, Kevin B and Nicholson. Bayesian artificial intelligence. cRc Press, 2003.

[25] Stefan Krauß. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. Technical report, Deutsches Zentrum für Luft- und Raumfahrt, Köln, 1998.

[26] Daeyeol Lee, Hyojung Seo, and Min Whan Jung. Neural Basis of Reinforcement Learning and Decision Making, 2012.

[27] Dar-Shyang Lee Dar-Shyang Lee and S.N. Srihari. A theory of classifier combination: the neural network approach. Proceedings of 3rd International Conference on Document Analysis and Recognition, 1, 1995.

[28] X Li and Q Ji. Active affective state detection and user assistance with dynamic Bayesian networks. Systems, Man and Cybernetics, Part A, 35(1):93–105, 2005.

[29] Joshua Lifton and Joseph A. Paradiso. Dual reality: Merging the real and virtual. In Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, volume 33 LNICST, pages 12–28, 2010.

[30] Joshua Harlan Lifton. Dual Reality : An Emerging Medium. PhD thesis, 2007.

[31] LN Long and Ankur Gupta. Scalable massively parallel artificial neural networks. Journal of Aerospace Computing, Information, and Communication, 5(1):1–11, 2008.

[32] M. Luber, GD Tipaldi, and KO Arras. Place-dependent people tracking. The International Journal of Robotics Research, 30(3):280–293, 2011.

[33] Matthias Luber, Luciano Spinello, Jens Silva, and Kai O. Arras. Socially-aware robot navigation: A learning approach. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, 2012.

[34] Matthias Luber and JA Stork. People tracking with human motion predictions from social forces. IEEE International Conference on Robotics and Automation, pages 464–469, 2010.

[35] Matthias Luber, Gian Diego Tipaldi, and Kai O. Arras. Better models for people tracking. In Proc. of the Int. Conf. on Robotics & Automation (ICRA), Shanghai, China, 2011.

[36] Jianchang Mao. Why artificial neural networks? Communications, 29:31–44, 1996.

[37] T A Marsland and Yngvi Björnsson. Variable-Depth Search. In ACG9, pages 9–24, 2001.

[38] George A Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review, 63(2):81–97, 1956.

[39] Onivola Henintsoa Minoarivelo. Application of Markov Decision Processes to the Control of a Traffic Intersection. Diploma, University of Barcelone Spain, 2009.

[40] Yongchan Na Yongchan Na and Jihoon Yang Jihoon Yang. Distributed Bayesian network structure learning. Industrial Electronics (ISIE), 2010 IEEE International Symposium on, 2010.

[41] Greg S Barnes Nelson. Real Time Decision Support: Creating a Flexible Architecture for Real Time Analytics. SUGI, 29(109), 2004.

[42] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Marcelo H. Ang Jr. and Oussama Khatib, editors, ISER, volume 21 of Springer Tracts in Advanced Robotics, pages 363–372. Springer, 2004.

[43] Buse Melis Ozyildirim and Mutlu Avci. Generalized classifier neural network. Neural Networks, 39:18–26, 2013.

[44] Prof. Jörg Hoffmann. Automatic Planning Lecture 1., 2014.

[45] A Robinson-Mosher and Christopher Egner. Learning Traffic Light Control Policies, 2005.

[46] Danko A. Roozemond. Using intelligent agents for urban traffic control control systems. In In Proceedings of the International Conference on Artificial Intelligence in Transportation Systems and Science, pages 69–79, 1999.

[47] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 3rd edition. Prentice Hall, 2009.

[48] WR Schwartz. Human detection using partial least squares analysis. Computer Vision, 2009 IEEE 12th, 2009.

[49] Herbert A Simon, Allan Newell, and J C Shaw. Elements of a Theory of Human Problem Solving. Psychological Review, 65(3), 1958.

[50] Herbert Alexander Simon. The New Science of Management Decision. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1977.

[51] Peter Stone and RS Sutton. Scaling reinforcement learning toward RoboCup soccer. ICML, (June):537–544, 2001.

[52] R S Sutton and A G Barto. Reinforcement learning: an introduction. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, 9:1054, 1998.

[53] Chiu-Che Tseng and Piotr J Gmytrasiewicz. Real Time Decision Support System for Portfolio Management. In HICSS, page 79, 2002.

[54] François van Lishout, Guillaume Maurice Jean-Bernard Chaslot, and Jos W H M Uiterwijk. Monte-Carlo Tree Search in Backgammon. In Proc. Comput. Games Workshop, pages 175–184, Amsterdam, Netherlands, 2007.

[55] Lucian Vintan and Arpad Gellert. Person movement prediction using neural networks. In First Workshop on Modeling and Retrieval of Context, 114(4):1–12, 2004.

[56] Z Wang, MP Deisenroth, and HB Amor. Probabilistic Modeling of Human Movements for Intention Inference. Proceedings of Robotics: Science and Systems, 32(2):1–8, 2012.

[57] NC Waugh and DA Norman. Primary memory. Psychological review, 72(2), 1965.

[58] Arturo Yee and Matías Alvarado. Pattern Recognition and Monte-CarloTree Search for Go Gaming Better Automation. In Advances in Artificial Intelligence–IBERAMIA 2012, pages 11–20, Cartagena de Indias, Colombia, 2012.

[59] Yang Zhang and Simon Fong. Real-time clinical decision support system with data stream mining. BioMed Research, 2012:580186, January 2012.

[60] Christine J Ziemer, Jodie M Plumert, James F Cremer, and Joseph K Kearney. Estimating distance in real and virtual environments: Does order make a difference? Attention, perception & psychophysics, 71:1095–1106, 2009.