Hochschule Reutlingen
Reutlingen University

INF
Informatik

Uwe Kloos, Natividad Martínez, Gabriela Tullius (Hrsg.)

Informatics Inside
connect(IT);

Informatik-Konferenz an der Hochschule Reutlingen
9. Mai 2018

9 783000 586453

Impressum

Anschrift:
Hochschule Reutlingen / Reutlingen University
Fakultät Informatik
Human-Centered Computing
Alteburgstraße 150
D-72762 Reutlingen

Telefon: +49 7121 / 271-4002
Telefax: +49 7121 / 271-4042

E-Mail: infoinside@reutlingen-university.de
Internet: http://infoinside.reutlingen-university.de

Organisationskomitee:
Prof. Dr. Gabriela Tullius, Hochschule Reutlingen
Prof. Dr. Natividad Martínez, Hochschule Reutlingen
Prof. Dr. Uwe Kloos, Hochschule Reutlingen

Benjamin Batt
Claudiu Bräuer
Sinem Cicek Celik
Emanuel Geiger
Frauke Grieble
Peter Grupp
Pia Laubacher
Öznur Öner
Katharina Pavic
Ngoc Linh Phan
Claudia Ryniak
Josia Scheytt
Christian Steinmann
Clemens Weißenberg
Vanessa Willenbrock
Steffen Wittig

Herstellung und Verlag: Hochschule Reutlingen
ISBN: 9 -83000-586453

Inhaltsverzeichnis

Longpaper

Social Crowd Simulationen zur Belebung virtueller Welten*

Steffen Wittig
Reutlingen University
Steffen.Wittig@Student.Reutlingen-University.DE

Abstract

Die Simulation menschlichen Gruppenverhaltens kann bei der Kapazitäten-, Risiko- und Evakuierungs-Planung von Gebäuden hilfreich sein, bei der Produktion von Filmen für eindrucksvolle Massen-Szenen eingesetzt werden oder virtuelle Schauplätze in Echtzeit-Anwendungen beleben. Die Herausforderungen liegen vor allem in einem realistischen Erscheinungsbild der virtuellen Crowd, glaubwürdigem Verhalten innerhalb eines sozialen Verbundes, realitätsnahen Animationen und der Wahrung der Echtzeitfähigkeit interaktiver Anwendungen. Im Rahmen dieser Arbeit wird der aktuelle Stand der Technik vorgestellt, Technologien evaluiert und ein Crowd Simulation Prototyp mit der Unity Engine implementiert.

Schlüsselwörter

Crowd Simulation, Künstliche Intelligenz, Sozialverhalten

CR-Kategorien

4.7 [**Real-time systems and embedded systems**]

1 Einleitung

Das Abbilden von menschlich anmutendem Verhalten auf eine große Menge virtueller Figuren (Agenten) wird als Crowd-Simulation bezeichnet. Das Verhalten der Agenten lässt sich in der Regel in zwei verschiedene Ansätze einteilen. Der einfachste Ansatz ist es, das Verhalten in Script-Form durch die Aneinanderreihung vieler expliziter (Low-Level) Aktionen zu programmieren. Hierbei sind autonomes Handeln und die Reaktion auf eine sich verändernde Umgebung des Agenten meist nur rudimentär umgesetzt und wird als *Programmed Crowds* bezeichnet. So lassen Sich einfache Simulationen erzeugen, bei denen jeder Agenten jeweils nur einen einzigen Zweck verfolgt, beispielsweise Fußgänger, die aus einem brennenden Gebäude fliehen. Ein anderer Ansatz ist die Entwicklung von komplexen autonomen Verhaltensabläufen, welche fortlaufend Feedback aus der Umgebung des Agenten verarbeiten, sogenannte *Autonomous Crowds*. Diese erlauben die Simulation einer großen Menge von virtuellen, selbstständig handelnden, menschlichen Agenten - benötigen aber mehr Ressourcen zur Berechnung des Verhaltens, als *Programmed Crowds*. [1], [2]

Die Anwendungen für autonome Crowd-Simulations sind vielfältig: Exakt simuliertes Gruppenverhalten kann bei der Kapazitäten-, Risiko- und Evakuierungs-Planung von Gebäuden hilfreich sein. Die Unterhaltungsindustrie kann mit besonders realistisch wirkenden, großen Mengen von virtuellen Charakteren eindrucksvolle Szenen für Filme und Spiele schaffen. Virtuelle Städte können durch Echtzeit 3D-Crowds bevölkert werden um lebhafter zu wirken. Die Herausforderungen liegen vor allem in einem realistischen Erscheinungsbild der virtuellen Crowd (Dichte,

---

*

Betreuer Hochschule: Prof. Dr. rer. nat. Uwe Kloos
Hochschule Reutlingen
Uwe.Kloos@Reutlingen-University.de
Informatics Inside 2018
Wissenschaftliche Vertiefungskonferenz
09. Mai 2018, Hochschule Reutlingen

Verteilung und Variation des Aussehens der einzelnen Agenten, ohne sie einzeln zu modellieren), nachvollziehbarem, real wirkendem Verhalten (zum Beispiel dem prediktivem gegenseitigen Ausweichen), glaubwürdiger Animation sowie dem Wahren der Echtzeitfähigkeit der Anwendung. [1]
Damit das Verhalten einer Gruppe von Agenten glaubwürdig wirkt, müssen ihre einzelnen Mitglieder bei näherer Betrachtung wie ein Teil eines sozialen Verbundes agieren, einer *Social Crowd*. Dies kann sich durch das Verfolgen verschiedener geselliger oder egoistischer Ziele äußern, die der Erfüllung von Bedürfnissen dienen. Damit die Agenten ihre virtuellen Ziele und Bedürfnisse erreichen können, müssen sie mit künstlicher Intelligenz ausgestattet werden. Sie müssen in der Lage sein, zu Ziel-Koordinaten im Dreidimensionalen Raum zu navigieren. In komplexen Umgebungen ist es jedoch nicht ausreichend, die Agenten auf geradem Weg auf die Ziel-Koordinaten zusteuern zu lassen, da nicht davon auszugehen ist, dass ein direkter Weg (Luftlinie) existiert und vorhandene Wege nicht durch dynamische Hindernisse versperrt sein könnten. Um nicht in Sackgassen zu enden oder ständig gegen Wände zu laufen, muss der Agent mit einem *Pathfindung*-Algorithmus einen gültigen Pfad zu seinem Ziel ermitteln.
In Echtzeit-Applikationen wird die Wegfindung zumeist durch manuell erzeugte oder automatisch generierte Wegpunkt-Netze erreicht. Wird ein möglichst effizienter Algorithmus auf diese Strukturen angewendet, können die virtuellen Agenten einen, je nach angewandtem Algorithmus, (annähernd) optimalen Pfad zu ihrem Ziel ermitteln.
In dieser Arbeit werden zunächst der Stand der Technik echtzeitfähiger Crowd-Simulationen vorgestellt und anschließend eine beispielhafte Implementierung sozial agierender Agenten beschrieben.

2 Stand der Technik

Die Simulation menschlicher Gruppen fußt auf Forschungsergebnissen der Simulation einfacher Entitäten, wie Vögel- und Fischschwärmen. 2001 stellten Musse und Thalmann [3] ein hierarchisches Modell zur Simulation virtueller menschlicher Crowds vor, bei dem Gruppen als intelligente Strukturen die einzelnen, darin befindlichen Agenten leiten. Eine Crowd setzt sich widerum aus einer Menge von Gruppen zusammen.

2.1 Echtzeitfähiges Gruppenverhalten

Um die Echtzeitfähigkeit zu wahren werden Entscheidungssysteme verwendet, die reaktive Verhaltensweisen (zum Beispiel Kollisionsvermeidung) implementieren, welche weniger rechenintensiv sind als rationale Entscheidungsmodelle. [3], [1]
Unterhaltungssoftware wie die Städtebau-Simulation *Cities Skylines*, oder Actionspielen wie *Grand Theft Auto V* verwenden einfache, echtzeitfähige Ansätze zur glaubwürdigen Visualisierung von Straßenzügen und öffentlichen Plätzen bei denen Personen den Anschein erwecken, sich ihrer Umgebung bewusst zu sein und sich zielgerichtet zu bewegen. Die Spielereihe *Die Sims* kann als unterhaltungsfokussierte Sozialverhaltens-Simulation angesehen werden und zeigt, wie Agenten auf in realitätsbasiert modellierten Umgebungen (Wohngebäude, öffentliche Plätze, Läden) ihre menschlichen Bedürfnisse erfüllen, zum Beispiel durch Interaktion mit anderen Agenten (Gespräche, zusammen Spielen, etc.).
Die Herausforderung bei der Entwicklung realitätsnah handelnder Passanten liegt in der Wegplanung, dem Vermeiden von Kollisionen mit anderen Passanten und der Echtzeitfähigkeit bei der Simulation großer Menschenmengen (> 100 Agenten). Daniel Thalmann beschäftigt sich seit den späten 1990er Jahren mit Crowd Simulations. Seine Forschungsergebnisse mit Branislav Ulicny ([2]) führten 2001 zur Echtzeit-Simulation von Notfallszenarien in virtuellen Städten mit unterschiedlich handelnden Agententypen (normale Bürger, Feuerwehrleute und Sanitäter).
Hierbei wurden die Verhaltensweisen jedes

einzelnen Agenten, basierend auf einem Regelwerk ausgewählt, welches persönliche Attribute der Agenten zur Wahl ihres konkreten Verhalten interpretiert. Verhaltensweisen wurden als verschachtelbare Finite State Machines implementiert, welche parallel ablaufende Teil-Verhalten erlauben. Neben Gruppenzugehörigkeiten und dem Vermeiden von Hindernissen, wurden zudem Interest Points (Lokalitäten, welche die Agenten passieren möchten) und Action Points (Lokalitäten, bei deren Erreichen die Agenten eine Aktion ausführen) implementiert. Das größte Problem beim Erreichen der Echtzeitfähigkeit der Simulation stellte die Visualisierung der Agenten dar. [2]

Verschiedene regelbasierte Verhaltensmodelle erzeugen realistische menschliche Bewegungsmuster für niedrig- bis mitteldichte Menschenmengen, berücksichtigen aber in der Regel aber nicht den Kontakt der einzelnen Agenten und können somit nicht das körperliche Drängen simulieren, welches sich in hochdichten Menschenmengen abspielt - Stattdessen wird der Kontakt und somit eine hohe Dichte vermieden, indem Warte-Regeln implementiert werden [4]. Studien an realen Menschenmengen zeigten, dass Menschen sich mit steigender Menschendichte sich stattdessen zunächst langsamer fortbewegen, bevor sie zum Stehen kommen [5].

Das Problem der Wegfindung und -planung wurde bereits im Rahmen der Robotik-Forschung umfassend gelöst. Unter dem Gesichtspunkt, kooperative Aufgaben zu lösen, wurden Multiagent Path Planning Lösungsansätze entwickelt, die verlässliche Ergebnisse liefern, deren Anwendung bei großer Agentenmassen wegen ihrer hohen Komplexität aber noch nicht echtzeitfähige Simulationen ermöglicht. Zu den bekanntesten Methoden zählen Social Force Models, Probabilistic Roadmaps, Visibility Graphs und Potential Fields. [1]

Helbing's Social Force Model ([6]) betrachtet jeden Agent als Partikel, der sozialen Kräften unterliegt, wodurch ein sich selbst organisierender Schwarm entsteht, der dem beobachteten Verhalten von Passanten nahe kommt. Visibility Graphs eignen sich für eine große Zahl virtueller Agenten: Hierbei werden nur Eckpunkte der beschreitbaren Umgebung miteinander verbunden, wenn diese voneinander gesehen werden können. So wird, inspiriert von Voronoi Diagrammen, automatisch die Topologie der Umgebung extrahiert und ein Navigationspfad berechnet werden. [1]

Das Vermeiden von Kollisionen bevor Sie entstehen (Local Avoidance) ist ein weiteres Problem echtzeitfähiger Anwendungen. Idealerweise kennt jeder Agent den zukünftigen Geschwindigkeitsvektor der Agenten um sich herum oder kommuniziert mit den anderen Agenten, um dann entweder seinen geplanten Weg zu ändern oder seinen aktuellen Geschwindigkeitsvektor anzupassen. Van den Berg et al. schlugen 2008 ([7]) das "reciprocal velocity obstacle" Konzept vor, bei dem Agenten auch ohne explizite Kommunikation mit anderen Agenten in dichtbevölkerten Umgebungen statische und bewegliche Objekte in Echtzeit vermieden. Hierbei berücksichtigen Agenten, dass andere Agenten ebenfalls ähnliche Ausweichmanöver einleiten würden. [1]

Potential Fields und das verwandte Zelluläre Automaten Modell (Cellular Automata Model), teilen die Umgebung der Agenten in Rasterzellen auf, welche unter anderem Informationen über die ideale Laufrichtung und die Agentendichte innerhalb der Zelle enthalten. Agenten bewegen sich von Zelle zu Zelle, wobei ein Weg mit möglichst geringer Agentendichte gefunden werden kann, was zu einer gleichmäßigeren Verteilung, aber bei hochdichten Mengen auch zu auffälligen Mustern (aufgrund der Bewegung entlang der Rasterzellen) führt. Zudem ist die ständige Aktualisierung der Rasterzellen rechenaufwändig. [4], [5]

Typischerweise wird jedoch ein Navigation Mesh verwendet um globale Routen durch eine virtuelle Umgebung zu planen. Während Agenten die geplante Route beschreiten finden Local Avoidance Routinen Anwendung, um anderen beweglichen Charakteren und dynamischen Hindernissen

auf der Route auszuweichen. [5]
Zur komfortablen Modellierung und zur Ermöglichung der Wiederverwendbarkeit von Teil-Verhalten von umfangreicheren Verhaltensmustern, werden bei proprietären Lösungen oft Entscheidungsbäume oder Zustandsmaschinen implementiert, da sie sich gut für die Entwicklung von Regelwerken eignen. Mit steigender Komplexität der Verhaltensmuster werden die hierfür häufig bereitgestellten GUIs für menschliche Benutzer allerdings auch zunehmend schwieriger zu erfassen und zu navigieren.

2.2 Soziale Aspekte

Thalmann beschreibt in seinem Beitrag *Crowd Simulations* im *Encyclopedia of Computer Graphics and Games* ([1]) umfassend die sozialen Aspekte von Crowd Simulations:
Die Spanne des Verhaltens von Menschen in Mengen reicht von völliger Gelassenheit bis zum Wahnsinn, was wiederum zu einem fröhlichen oder verzweifelten Gefühlszustand der einzelnen Simulationsteilnehmer führt. Eine weit verbreitete Annahme ist, dass sich die Persönlichkeit von Menschen in Massensituationen temporär ändert, wenn sie sich einer Gruppe anschließen. Viele Autoren aus dem Bereich der Massenpsychologie stellen fest, dass die bezeichnende Eigenschaft von Massensituationen das Entfallen von üblicherweise anwendbaren kulturellen Regeln, Normen und Ordnungsformen ist. So entfalle in Paniksituationen beispielsweise die Regel Schlangen zu bilden, wenn auf etwas gewartet wird. [1]
Das Gesamtbild eines virtuellen Massenverhaltens lässt sich, so Musse und Thalmann ([8]), durch die Wechselbeziehungen der darin befindlichen Gruppen beschreiben. Gruppen bilden sich bei der Begegnung von virtuellen Agenten, bei denen Sie Ihre emotionalen Parameter (beispielsweise der emotionale Status) und Ziele (Interest und Action Points) vergleichen und sich bei großer Ähnlichkeit zusammenschließen. Innerhalb der Gruppen werden Wegpunkte und Ziele geteilt, sodass die Agenten sich nicht mehr weit voneinander entfernen. Treffen sich zwei Gruppen mit ähnlichen Attributen kann ein Gruppenübergreifender Dominanz-Wert entscheiden, welche Gruppe sich der anderen anschließt. [8]
Pelechano, Allbeck, and Badler ([4]) stellten 2007 eine Lösung für hochdichte Crowd Simulations vor, bei der jeder Agent über Persönlichkeits-Attribute (beispielsweise geduldig, panisch) und eine Influence Disk verfügt - eine Region vor dem Agenten, welche Warte-Verhalten auslöst, wenn ein anderer Agent sich in den Radius der Influence Disc bewegt. Während geduldige Agenten warten, wird ein Agent mit ungeduldigem Attribut den Artgenossen im Weg eher ignorieren ihn aus dem Weg drücken. [4]
Die sozialen Aspekte einer Crowd Simulation sind also in Persönlichkeitsattributen, Gruppenbildung und Local Avoidance, sowie in der Kommunikation von Agenten untereinander zu suchen.
Weitere Aspekte, welche die glaubwürdigen Simulation von öffentlichen Flächen unterstützen könnten, sind wissens-sammelnde Agenten, die Informationen über ihre Umwelt (Points of Interest) erst durch das Erfassen von Wegweisern, den Wissensaustausch mit anderen Agenten oder Sichtkontakt während ungezwungenem Erkunden erfahren. Dies gibt Agenten eine zusätzliche Motivation, sich mit anderen Agenten auszutauschen. Gleichzeitig lassen Sie sich über ein Leitsystem aus Schildern, abweichend vom kürzesten Weg, lenken um so beispielsweise die Fußgängerdichte an bestimmten Lokalitäten zu steuern.
Agenten könnten neben Persönlichkeitsattributen auch mit Bedürfnissen ausgestattet werden, wie Hunger, Durst oder Harndrang, sowie der Wunsch nach einem Informationen, einem Gespräch, Musik, oder bestimmten Waren, deren Dringlichkeit über die Laufzeit der Simulation stets steigt. Points of Interest könnten der Befriedigung (=Verringerung der Dringlichkeit) dieser Bedürfnisse dienen,wie Schaufenster oder sogar bestimmte Artikel in einem Schaufen-

ster (je nach Granularität der Szene), Waren in einem Laden, Sehenswürdigkeiten, Plätze/Treffpunkte, Sitzgelegenheiten, Toiletten, andere Agenten oder Schilder.
Verschiedene Software-Lösungen ermöglichen die Simulation von Crowds: Das Plugin *Golaem* für *Autodesk Maya*, welches visuell ansprechende Menschenmengen erzeugen kann, allerdings nicht echtzeitfähig ist. CrowdMaster ist ein Blender-Addon, welches die Kernfunktionalität von Golaem nachahmt, dabei allerdings nicht dessen Umfang und Benutzerfreundlichkeit erreicht. *Viswalk* von *PTV* erlaubt die Simulation von Fußgängern in Echtzeit und stellt die Agenten wahlweise zweidimensional als Punkte oder dreidimensional als rudimentär animierte Menschenmodelle dar, die allerdings keine Kenntnis voneinander nehmen, wodurch nicht mehr als eine Fluß-Simulation.

3 „Social Crowd“ Prototyp

Zentraler Bestandteil dieser wissenschaftlichen Vertiefungsthematik ist ein Prototyp, der mit der Unity Engine implementiert wurde. Durch einen modularen Aufbau soll die mögliche Weiterentwicklung erleichtert werden.
Zunächst wurden die verfügbare Features und Plugins für Unity evaluiert, um eine technologische Ausgangsbasis für den angestrebten Prototypen einer sozialen Crowd Simulation zu finden. Nach der Evaluationsphase folgte die Implementierung der sozialen Agenten, wobei zentrale Anforderungen die glaubwürdige Wegfindung und rudimentäre *Local Avoidance* waren.

3.1 Technologiewahl

Betrachtet wurden die internen AI Features der Unity Engine, das RAIN AI Plugin und das A* Pathfinding Project Plugin.
Unity bietet im **UnityEngine.AI** Package Methoden zur Lösung von Pfadfindungs-Problemen. Andere Aspekte künstlicher Intelligenz werden nicht abgedeckt. Die Pfadfindung in Unity basiert ausschließlich auf NavMeshes und Off Mesh Links, welche disjunkte Flächen des NavMesh miteinander verbinden können (zum Überspringen von Abgründen oder dem Erklimmen von Hindernissen). Ein NavMesh bildete komplexe Szenen-Geometrie auf ein Mesh mit wenigen Vertices ab, das nur die betretbare Fläche beinhaltet und so wenig rechenintensiv Pfadfindungsprobleme lösen kann. Es lassen sich in Unity unterschiedliche Navigationsebenen mit unterschiedlichen Traversierungskosten definieren (denkbar sind Straße, Matsch, Schotter, etc.).
Zentrale Aspekte werden über ein eigenes GUI-Panel abgewickelt. Hier lassen sich u.a. verschiedene Agenten-Typen definieren, die sich durch ihre Höhe und Radius, sowie ihr Stufen- und Hang-Verhalten unterscheiden, die unterschiedlichen Geländetypen und ihre Traversierungskosten definieren. Ausgewählte Objekt können als statisch definiert werden - sie werden dann beim Backen von Navmeshes berücksichtigt. Anschließen kann hier auch das Navmesh generiert werden, was bei jeder Änderung der Szenerie im Editor wiederholt werden muss.
Die Navigation und Fortbewegung des Agenten werden über den allgemeinen *CharacterController Component* und einen *NavMeshAgent Component* realisiert. Während ein *GameObject* durch den NavMeshAgent Component fortbewegt wird, werden keine Kollisionen des Agenten im Rahmen der Physik Engine von Unity berücksichtigt (*Collider Componenten* sind wirkungslos): Der Agent wird Collider, die sich dynamisch im NavMesh bewegen, reaktionslos durchlaufen.
Hindernisse, die nicht statisch sind, müssen einen Nav MeshObstacle Component erhalten, damit ein Agent sie bei seiner Wegplanung berücksichtigt. Alternativ können sie auch dynamisch aus dem NavMesh ausgeschnitten werden, damit Agenten sie bei ihrer Wegfindung erkennen um alternative Routen zu suchen.
Das populäre Plugin ***A* Pathfinding Project*** aus dem Unity Asset Store beschreibt sein Autor selbst als besonders schnell und einfach zu nutzen. Es unterstützt manuelle

und automatisch generierte NavMeshes und Wegpunkt-Netze (Point Graphs), sowie automatisch generierte Grids.
Eine kostenlose Version des Plugins bietet zwar die notwendigsten Features, NavMeshes können aber nur mit der Pro-Version automatisch generiert werden. Im Gegensatz zum Unity AI Package, bietet das Plugin selbst keine Methoden zur Fortbewegung der Agenten oder zur Programmierung ihres Verhaltens an, sondern beschränkt sich auf Methoden zur Pfadfindung. Die Traversierung des resultierenden Arrays mit Wegpunkten des Pfadfindungsprozesses muss eigenhändig implementiert werden.
Um das A* Pathfinding in eine Szene zu integrieren, muss einem Objekt der *AstarPath Component* hinzugefügt werden. In dessen Konfigurationsdialog werden NavMeshes hinzugefügt und andere rudimentäre Einstellungen vorgenommen. Der Agent benötigt einen *Seeker Component* um Wege zu berechnen und einen weiteren händisch zu implementierenden *Component*, welche die Berechnung des Weges anstößt und den errechneten Weg in eine Bewegung umsetzt.
Alle Klassen des A* Pathfinding Project werden als quelloffene wohldokumentierte C# Klassen in das Projekt integriert.
Das ***RAIN AI*** Plugin erweitert Unity um eine Oberfläche zur Entwicklung von *Behavior Trees*. Unter einem Root-Knoten können hier *Action*- oder *Decision*-Knoten hinzugefügt und beliebig verschachtelt werden um komplexe Verhalten zu programmieren.
Im einfachsten Fall befindet sich unter dem Root-Knoten eine *Move* Action, welche dem Agenten ein Ziel (Objekt mit NavigationTargetRig Component) sowie eine Bewegungsgeschwindigkeit zuordnet und die Wegfindung und Navigation des Agenten über den Unity *CharacterController Component* veranlasst
Die Wegfindung findet entweder über ein Wegpunkte Netz oder ein automatisch generiertes NavMesh und den *NavMeshRig Component* statt, welcher mittels Unity-Ebenen-Masken begehbare Umgebung, Hindernisse und Agenten unterscheidet. Während der Navigation und Fortbewegung werden, wie bei den Unity AI Packages, keine Kollisionen zwischen Collidern der Agenten und der Umgebung berücksichtigt. Die Agenten können mittels visuellen und auditiven Sensoren andere Entitäten (*EntitiyRig Component*) wahrnehmen. Entitäten können wiederum visuelle und auditive *Aspects* zugeordnet werden, welche schließlich in einem *Detect* Knoten im Behavior Tree wahrgenommen werden. Variablen, welche dem Behavior Tree zur Verfügung stehen sollen, werden im *Memory* Tab des Agenten definiert.
Mittels *Custom Actions* können die Fähigkeiten der Agenten über die bereits integrierten Actions hinaus erweitert werden. Custom Actions sind C# Klassen, welche abstrakte Methoden der RAINAction Klasse implementieren müssen und auf die üblichen Unity API Klassen zugreifen können. Im Zusammenspiel mit den Variablen im Memory des Agenten können so auch komplexe Verhaltensweisen entwickelt werden. Ein Vorteil ist die Wiederverwendbarkeit der Custom Actions in verschiedenen Behavior Trees.

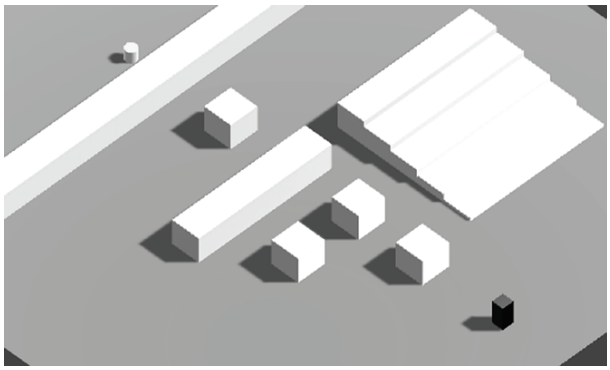

Abbildung 1: Test-Szenario Start Zustand. Der helle Zylinder oben links ist das Navigationsziel und der dunkle Quader unten rechts ein Agent.

Für einen Vergleich der Echtzeitfähigkeit der Lösungen wurde ein einfaches Testszenario entwickelt in dem mittels Mauseingaben Agenten erzeugt und ihr Navigationsziel neu gesetzt werden kann. Hierbei zeigte sich bei 100 zeitgleich aktiven Agenten, dass die

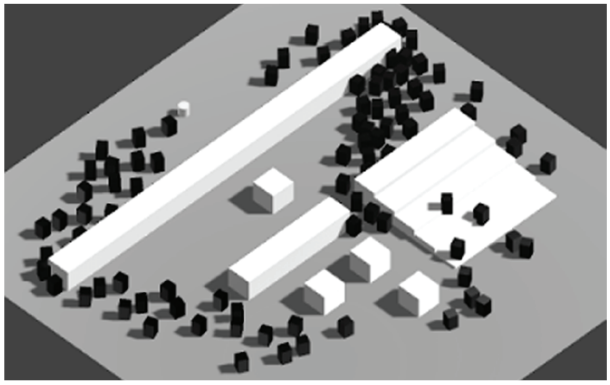

Abbildung 2: 91 Agenten versuchen das Navigationsziel im oberen linken Quadranten der Abbildung zu erreichen.

Framerate unter Verwendung des RAIN AI Plugin auf 24 FPS einbricht, während die Unity AI Klassen und A* Pathfinding noch ohne merkliche Leistungseinbrüche 60 FPS erreichten. Bei 500 Agenten lief letztendlich nur das Szenario mit den Unity AI Klassen noch flüssig, während RAIN AI bei 4 FPS und A* Pathfinding nur noch 10 bis 15 FPS erreichten.
Die visuelle Programmierung der Verhaltensbäume mit den Mitteln des RAIN AI Plugins ist für simple Spiele-AI Routinen, die intelligent navigierende Agenten benötigen, geeignet. Bei komplexen Verhaltensweisen, wie sie im Kontext einer sozialen Simulation zu erwarten sind, ist jedoch mit unübersichtlich großen Bäumen zu rechnen, welche in einer Benutzeroberfläche nur noch schwer nachvollzogen und verwaltet werden können.
Nachteilig für die sozialen Crowd Simulation ist bei allen vorgestellten Lösungen das Beschreiten des kürzesten Weges, und somit die Tendenz, dass Agenten sich nahe an Wänden bewegen und Kurven schneiden, sowie die “übernatürliche” Kenntnis aller Agenten über die gesamte Geographie der Szene, ohne sie zuvor beschritten oder Wegweiser/Karten gelesen zu haben.
Die Wahl zur Implementierung des Prototypen fiel auf die Navigationsklassen von Unity, da Sie sich als performante Lösung herausstellten. Das Fehlen von weiteren KI Hilfsmittel machte es allerdings nötig, die Implementierung aller anderen Aspekte der Agenten-Intelligenz selbst vorzunehmen.

3.2 Implementierung

Ein Prototyp, welcher es ermöglicht, in einer einfachen Testumgebung (Abbildung 1) eine große Menge Agenten (Abbildung 2) zu initialisieren und Ihnen ein Ziel zuzuweisen, wurde implementiert. Agenten planen stets den kürzesten Weg zum Ziel, weichen einander aus und versuchen dabei stets einen Mindestabstand zu Wänden einzuhalten, außer sie finden nicht genug Platz bei einem Ausweichmanöver - dies wurde mittels höheren Traversierungskosten für Randbereiche des NavMeshes gelöst (Abbildung 5). Agenten suchen sich zufällig ein neues Ziel aus einer Menge von *Points of Interest*, wenn sie ihr zuletzt gesetztes Ziel erreicht haben. Agenten können unterschiedlich Aggressiv ausgeprägt sein, was sich auf Ausweichmanöver auswirkt. (Abbildung 3).

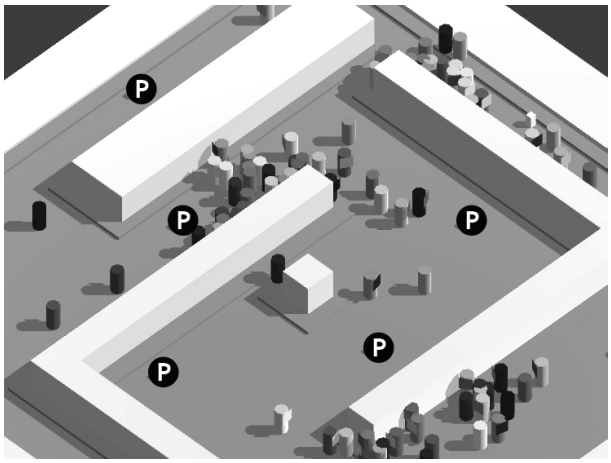

Abbildung 3: Die finale Social Crowd Simulation mit 101 Agenten (eingefärbt nach ihrem Aggressionswert: je dunkler, desto aggressiver). Sie besuchen alle in zufälliger Reihenfolge Points of Interest (mit P gekennzeichnet)

Um auf möglichst viele Szenarien anwendbar zu sein, wurde das Verhalten der sozialen Agenten modular implementiert: Hierzu wird, inspiriert von RAIN AI und der Architektur robotischer Systeme, die Interaktion des Agenten mit der Umgebung durch Sensoren und Aktoren umgesetzt.

Ein „Brain“-Modul fragt das Umgebungs-Feedback der Sensoren ab und steuert damit die Aktoren. Die visuelle und physikalische Repräsentation des Agenten wurde mit den bereits vorhandenen Komponenten der Unity Engine realisiert.
Kernstück der Implementierung sind der *Entity-Component*, der vom Szenario Entwickler jedem Objekt, das an der Simulation teilnimmt, hinzugefügt werden muss und der *Agent-Component*, der jedem Agenten hinzugefügt werden muss. Hindernisse, Points of Interest, sowie Agenten verwenden einen Entity-Component. Dieser besitzt keine weitere Intelligenz - sein einziger Zweck ist es, als Teil der Simulation von Sensoren wahrgenommen zu werden. Der Agent-Component dient als Container von Sensor- und Aktor-Komponenten, sowie dem Brain-Komponent, sowie der Attribute “aggressiveness” und “extroversion”, zwei Float-Werte, die zufällig für jeden Agenten festgelegt und innerhalb seiner Brain-Logik interpretiert werden um das Verhalten bei der Begegnung anderer Agenten zu steuern. Ein Attribut zur Speicherung der Gruppenzugehörigkeit ist vorgesehen, die Gruppen-Logik wurde aber nicht mehr implementiert. Damit Agenten sich durch Sensoren wahrnehmen, erbt der Agent-Component vom Entity-Component.
Zwei Sensor-Klassen ermitteln entweder alle Entities in einem bestimmten Radius um den Agenten (BasicSensor) oder senden Strahlen in einem bestimmten Winkelbereich vor dem Agenten aus (RadialSensor, siehe Abbildung 4). Ein „BasicMovement“-Aktor wurde implementiert, welcher dazu dient, zu einem Ziel-Punkt zu navigieren. Hierfür berechnet er sporadisch den Weg neu, um Hindernisse (andere Agenten, die von den Sensoren erfasst wurden) zu umgehen.
Die abstrakte Brain-Klasse beinhaltet die nötige Logik um einer periodischen Entscheidungs-Schleife auszuführen. Die SocialBrain-Klasse erweitert die Brain-Klasse um Methoden zur Auswertung der Sensor-Ergebnisse und zur Steuerung der Aktoren. Außerdem werden Gizmos

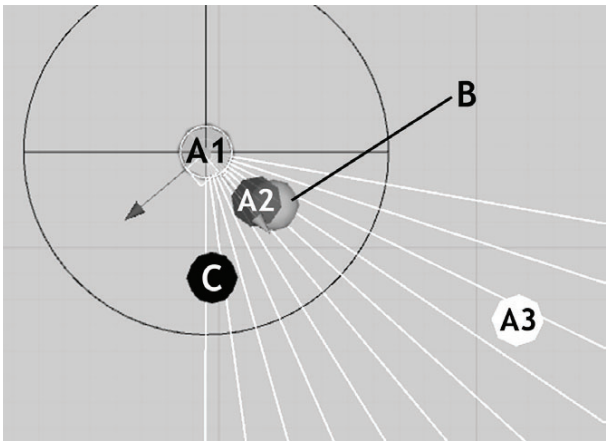

Abbildung 4: Visualisierung der Sensor-Wahrnehmungsmethodik des Agenten A1 im Unity Editor. Schwarzer Kreis: BasicSensor (Personal Space); Weiße Linien: RadialSensor (Vision). Die zuletzt wahrgenommene Position von A2 wird mit Punkt B dargestellt. Punkt C zeigt die ermittelten Ausweichkoordinaten um eine Kollision mit A2 zu vermeiden.

(Sensor-Bereiche, Navigationsziel und erkannte Agenten) im Editor-Fenster von Unity gezeichnet um das Verhalten der Agenten während der Entwicklung nachvollziehbarer zu gestalten (zu sehen in Abbildungen 6 und 4).
Mit zunehmender Anzahl und Nähe von Agenten im BasicSensor wird die Fortbewegungsgeschwindigkeit verringert, welche außerdem noch von den Aggressiveness- und Extroversion-Attributen des Agenten beeinflusst wird.
Werden andere Agenten mit dem RadialSensor wahrgenommen, prüft der Agent ob diese Ihm bereits ausweichen (Kommunikation von Agenten untereinander). Ist dies nicht der Fall, leitet er selbst ein Ausweichmanöver ein, indem ein neues Navigationsziel an einen nahegelegenen, freien Platz gesetzt wird. Das Extroversion-Attribut entscheidet darüber, wie ausladend das Ausweich-Ziel gewählt wird (introvertierte Agenten nehmen größere Umwege). Wird kein freier Platz zum Ausweichen gefunden, stoppt ein Agent. Angenten mit sehr hohem Aggressiveness-Attribut (>0.95) weichen nie

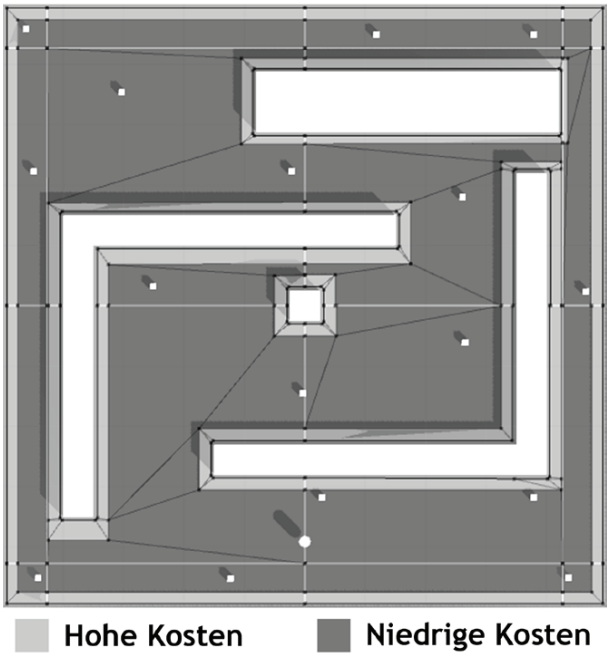

Abbildung 5: Visualisierung des gebackenen NavMeshes mit Traversierungskosten

aus und treffen so häufig mit anderen aggressiven Agenten zusammen.
Befindet sich ein Agent nahe seinem Ziel (etwa ½ Agenten-Durchmesser entfernt), wird das Ziel als erreicht angesehen und "vergessen". Der Agent hat kein Ziel mehr und bleibt stehen.
Der SocialCrowdController ist ein Hilfskonstrukt, das verhindert, dass die Simulation zum Erliegen kommt, wenn alle Agenten ihr Ziel erreicht haben. Dieser Component prüft in jeder Update-Schleife, ob sich Agenten ohne Ziel in der Simulation befinden und weist diesen einen zufällig ausgewählten Point of Interest als neues Ziel zu.

4 Fazit und Ausblick

Der entwickelte Prototyp erlaubte es, die Schwierigkeiten bei der Entwicklung sozialer Agenten zu erkennen. Die entwickelten autonomen Agenten nehmen, abhängig von ihren Attributen, aufeinander Rücksicht und weichen in den meisten Fällen aus um Kollisionen zu vermeiden. Die Verwendung eines NavMeshes mit höherer Gewichtung der Randbereiche verhindert, dass Agenten sich ständig an Wänden entlang schieben. So konnten die Anforderungen der glaubwürdige Wegfindung und rudimentären *Local Avoidance* erfüllt werden.
Auf dem Test-System lief die Simulation auch mit über 200 Agenten einer Bildwiederholrate von über 60 FPS. Eine Menge von 100 Agenten verstopfte allerdings schnell schmale Durchgänge. Bei längerer Laufzeit bilden sich in den engen Randbereichen des Szenarios Menschentrauben, vor denen Agenten ratlos stehen bleiben, da sie nicht die Fähigkeit besitzen, größere Umwege zu planen.
Als Weiterentwicklungsmöglichkeiten sind weitläufigere und realitätsnahere Testszenarios denkbar (Fußgängerzone einer Innenstadt oder ein Einkaufszentrum mit Läden). Zudem könnten Agenten mit Bedürfnissen ausgestattet werden, welche durch den Besuch von Points of Interest befriedigt werden - Agenten wählen dann als nächstes den Point of Interest, der ihr am wenigsten befriedigtes Bedürfnis erfüllt, beispielsweise einen Imbissstandstand, Foto-Laden oder Toiletten - so könnte Konsumentenverhalten simuliert werden.
Mit zunehmendem grafischen Realismus der Umgebung könnte der Einsatz von Computer Vision Fähigkeiten für Agenten erlauben zu ermitteln, wie gut Schildern in Bauwerken noch bei hoher Menschendichte wahrgenommen werden können (beispielsweise in Fluchtsituationen). Wegen der erhöhten Rechenlast der Bild-Interpretations-Algorithmen können möglicherweise nur einzelnen Agenten der Simulation die Umgebung auf diese Weise wahrnehmen.
Während der Entwicklung des Prototypen wurden geeignete Schwellwerte für Ausweichmanöver, Agenten-Geschwindigkeit und die Auswirkungen der *Extroversion*- und *Aggressiveness*-Attribute durch manuelle Versuche ermittelt und justiert bis ein möglichst glaubwürdig erscheinendes Ergebnis erreicht wurde. Um die Auswirkungen der Änderung dieser Attribute und gegebenenfalls weiterer Simulationsparameter der Simulation besser erkennbar zu machen, könnte die Manipulation über ein Benutzerinterface

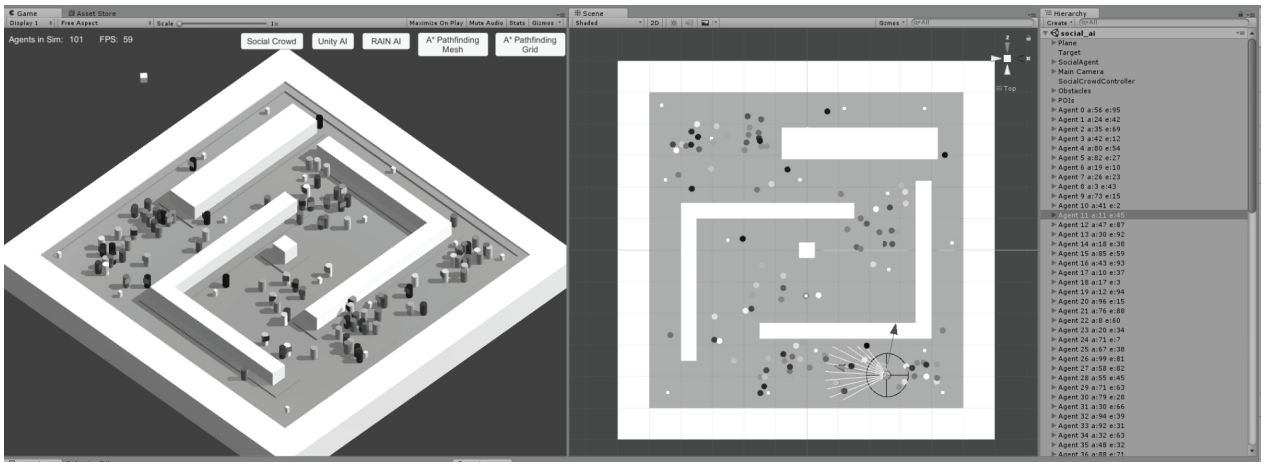

Abbildung 6: Ansicht der Social Crowd Simulation im Unity Editor. Das linke Panel zeigt die isometrische Ansicht, in welcher der Benutzer mit der Simulation interagieren kann. Das mittlere Panel zeigt eine Top-Down View mit Debug-Informationen in Form von Gizmos (für den ausgewählten Agenten). Im rechten Panel ist die Liste der GameObjects zu sehen, darunter viele der Agenten.

(z.B. durch Eingabefelder und Slider) zur Laufzeit der Simulation implementiert werden.

Literatur

[1] Daniel Thalmann. Crowd simulation. In Newton Lee, editor, *Encyclopedia of Computer Graphics and Games*, pages 1–8. Springer International Publishing, Switzerland, 2015-.

[2] Branislav Ulicny and Daniel Thalmann. Crowd simulation for interactive virtual environments and vr training systems. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2–3, 2001*, pages 163–170. Springer Vienna, Vienna, 2001.

[3] S. R. Musse and D. Thalmann. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[4] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[5] Wouter G. van Toll, Atlas F. Cook, and Roland Geraerts. Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds*, 23(1):59–69, 2012.

[6] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[7] Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, and Ming Lin. Interactive navigation of multiple agents in crowded environments. In Eric Haines and Morgan McGuire, editors, *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, page 139, New York, N.Y., 2008. ACM Press.

[8] Soraia R. Musse and Daniel Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Computer animation and simulation*, volume 97, pages 39–51, 1997.