公立はこだて未来大学 2025 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2025 Systems Information Science Practice
Group Report

プロジェクト名

数理モデリングプロジェクト

Project Name

Mathematical Modeling Project

プロジェクト番号/Project No.

09

プロジェクトリーダ/Project Leader

菊地皓太　Kota Kikuchi

プロジェクトメンバ/Group Member

菊地皓太　Kota Kikuchi
西尾柊太　Shuta Nishio
坂井康大　Kodai Sakai
櫻井孔士　Koushi Sakurai
島津諒雅　Ryoga Shimazu
鈴木創太　Sota Suzuki

指導教員

石尾隆　加納剛史　寺井あすか

Advisor

Takashi Ishio　Takeshi Kano　Asuka Terai

提出日

2026 年 1 月 21 日

Date of Submission

January 21, 2026

概要

現在，公立はこだて未来大学の食堂内ではピーク時における混雑の集中や，待ち行列の長大化による動線の交錯といった人流に関わる課題が表面化している．これらの課題は利用者の待ち時間増加や移動の制約など，利便性を低下させる要因となっている．そこで，本プロジェクトでは数理モデルを用いた人流シミュレータの作成により，本学の食堂における混雑緩和策の検証および提案を行った．

そのため，まず現状を把握するため，食堂レイアウトの測量ならびに人流の計測・観察を行った．加えて，利用者および従業員を対象としたアンケート・インタビューを実施し，不便を感じる場面や混雑が発生する要因を多角的に把握した．これらの調査を通して得られた実データを基に，食堂内の人流を対象として数理モデリングを行い，個々の人同士の相互作用や障害物回避，目的地志向の移動などが表現可能なエージェントベースドモデルである Social Force Model に基づくシミュレータを作成した．実環境での試行には時間的・人的コストが伴う一方で，本手法を用いた仮想環境上での反復的評価というアプローチは，複数案を再現性の高い条件下で比較や検討が可能であるという利点がある．

作成したシミュレータでは，ピーク時の利用状況を再現しつつ，動線設計や運用方法の変更が混雑状況に与える影響の比較検証を可能とした．さらに，混雑が発生する原因に関する仮説を立案し，シミュレーション結果に基づいて混雑緩和策の効果検証および改善提案を行った．本プロジェクトの成果は，公立はこだて未来大学の食堂における混雑緩和ならびに利便性向上に寄与する有意義な知見を提供するものである．

（※文責: 菊地皓太）

キーワード　数理モデリング，人流，Social Force Model，待ち行列，混雑，シミュレーション，マルチエージェント

（※文責: 菊地皓太）

Abstract

Currently, Future University Hakodate's cafeteria is experiencing issues related to people flow, such as congestion during peak hours and the intersection of traffic lines due to long waiting lines. These issues are factors that reduce convenience, such as increased waiting times for users and restrictions on movement. Therefore, in this project, we created a people flow simulator using a mathematical model to verify and propose measures to alleviate congestion in the university's cafeteria.
To understand the current situation, this project first surveyed the cafeteria layout and measured and observed people flow. In addition, we conducted questionnaires and interviews with users and employees to gain a multifaceted understanding of inconveniences and factors that cause congestion. Based on the actual data obtained through these surveys, we performed mathematical modeling of people flow in the cafeteria and created a simulator based on the Social Force Model, an agent-based model that can represent interactions between individuals, obstacle avoidance, and destination-oriented movement. While trials in a real environment require time and human resources, the iterative evaluation approach in a virtual environment using this method has the advantage of allowing multiple proposals to be compared and examined under highly reproducible conditions. The simulator we created reproduced peak usage conditions, and made it possible to comparatively verify the impact that changes to traffic flow design and operational methods had on congestion conditions. Furthermore, we formulated hypotheses regarding the causes of congestion, and based on the simulation results, verified the effectiveness of congestion mitigation measures and proposed improvements. The results of this project will provide meaningful knowledge that will contribute to mitigating congestion and improving convenience at the cafeteria of Future University-Hakodate.

（※文責: 菊地皓太）

Keyword Mathematical Modeling, People Flow, Social Force Model, Queueing, Congestion, Simulation, Multi-agent

（※文責: 菊地皓太）

目次

第 1 章　はじめに

本プロジェクトでは公立はこだて未来大学の食堂における混雑緩和策の検証および提案を行うことを目的に活動する．

（※文責: 菊地皓太）

1.1　背景

人の混雑は，日常生活の様々な場面で見られる．例えば，イベント会場や駅，商業施設，観光地などでは，特定の時間帯に人が集中し，移動や行動に支障をきたす可能性がある．こうした混雑は，利用者の待ち時間増加や移動の制約など，利便性を低下させるだけでなく，転倒や接触事故のリスク増加など安全面にも影響を及ぼす [1][2]．さらに，運営側にとっても誘導や案内，サービス提供の負荷が増大し，施設全体の運用効率の低下につながる．そのため，人流を適切に把握した上で，混雑の発生要因を特定し，状況に応じて緩和・制御するための方策を検討することが求められる．

大学の食堂も，こうした混雑が生じる場の一つである．大学の食堂は，学生や教職員に日常的な食事を提供する重要なインフラであり，とりわけ昼食時などのピーク時間帯には，多くの利用者が限られた時間に集中する傾向がある．これは利用者にとって時間的・心理的な負担となり，混雑を避けるために利用自体を控えるといった行動にもつながる可能性がある [3]．そのため，大学の食堂における人流を分析し，状況に応じた改善策を検討することは，効率的かつ円滑な運営を実現する上で重要である．

（※文責: 菊地皓太）

1.2　課題

現在の食堂では，昼休みなどのピーク時間帯に利用者が短時間に集中する傾向があり，その結果として混雑の発生や待ち行列の長大化といった問題が顕在化している．特に，配膳口やレジ周辺では利用者が滞留しやすく，動線が交錯することで移動が阻害されるなど，人流に起因する課題が多く見られる．これにより，通行のしづらさや安全性の低下といった問題も生じており，食堂全体の円滑な運用に影響を及ぼしている．また，これらの混雑は利用者の待ち時間を増加させるだけでなく，移動の自由度を制限し，食堂利用時の満足度や利便性を低下させる要因となっている．

以上のことから，ピーク時における人流の状況を把握し，待ち行列の発生位置や滞留が生じる原因を明確化することが求められる．その上で，動線設計や配膳，運用方法の改善など，混雑緩和に向けた具体的な対策を検討・実施していくことが不可欠である．特に，実環境での試行が困難な場合においても検証可能な手法を用いることで，効率的かつ合理的な改善策の立案が期待される．

（※文責: 菊地皓太）

1.3 到達目標

上記の課題を解決するため，食堂レイアウトの測量，人流の動向や待ち行列の長さ，滞在時間などを計測するとともに，利用者や従業員を対象としたアンケート，インタビューによる調査を実施した．これらの調査を通して得られた実データを基に，食堂内の人流を対象として数理モデルによる挙動の定式化を行い，個々のエージェント間の相互作用や障害物回避，目的地志向の移動などが表現可能である Social Force Model に基づくシミュレータを作成した．これにより，実際の混雑状況や人流を仮想空間上で再現し，時間変化に伴う可視化が可能となる．特に，ピーク時にどの地点で滞留が発生しやすいか，行列がどのように伸長するか，通路のボトルネックがどこに生じるかといった点を，把握することが可能である．

また，本シミュレータは，座席配置や待機列の形成位置，通路幅など，食堂内レイアウトを変更可能な設計とする．加えて，エージェントの移動速度，目的地の選択傾向，待ち行列の並び方などの行動特性についても，パラメータとして設定や変更を可能にする．これにより，現状の再現だけでなく，改善案を適用した場合の混雑緩和効果を比較検証することが可能となる．その分析結果に基づき，レイアウト変更や動線設計など，実現可能性の高い混雑緩和策の評価や提案を行う．

（※文責: 菊地皓太）

第 2 章　理論背景と先行研究および知識修得

本章では，本プロジェクトの背景として関連する数理モデルの一般的な考え方および，直接使用していないものも含めた先行研究を整理し，研究の着想や位置づけを踏まえて修得した知識について述べる．

（※文責: 坂井康大）

2.1　数理モデリングとは

数理モデリングとは，実験や観察から示唆される法則や仮定から，対象とする現象を数学の問題として定式化することである [4]．これは単なる抽象化作業に留まらず，現実世界で生じている社会的・工学的課題に対して，数量的な分析に基づいた理解と予測を可能にする点に重要な意義がある．すなわち，現象の背後にある因果関係を整理し，その仕組みを数式・モデル・シミュレーションなどの形で表現することで，「なぜそうなるのか」「今後どうなるのか」「改善策は何か」といった問いに客観的な根拠を持って解決に向けた道筋を立てることを目的とする．

以下に，数理モデルを作成する一般的な手順と，その手順において本プロジェクトで行った内容について述べる．

（※文責: 坂井康大）

2.1.1　数理モデリングの手順

本節では，本研究で用いた数理モデリングの一般的な手順を整理する．

第一に，数理モデリングは，現実世界に存在する問題意識を明確にする段階から始まる．この工程では，対象とする状況を観察し，問題が「どこで」「いつ」「どのように」発生しているのかを把握し，課題を定義することが重要である．そのため，対象空間の構造把握，関係する要素の整理，数量的・質的データの収集などを通じて，現象の特徴を明確化する．これらの観察結果は，後のモデル設計における制約条件や境界条件の設定に寄与する．

第二に，収集したデータや観察結果を踏まえ，問題の原因に関する仮説を構築する．現象が生じる要因について，対象要素の相互関係や振る舞いに着目し，「どの要因がどのように影響しているか」を整理する．例えば，行動の順序や配置条件，個体間の相互作用などが現象に影響を及ぼしている可能性を仮定し，条件の変化が結果に与える影響についての仮説を設定する．この工程は，モデル化の方向性を定める役割を担い，仮説の妥当性は後続の分析を通じて検証される．

第三に，設定した仮説を数理モデルとして表現する．この段階では，対象とする現象の性質に応じて，適切なモデル表現を選択する必要がある．一般に，個々の要素の振る舞いを記述するモデルや，全体の平均的な性質を扱うモデルなど，さまざまな数理的枠組みが用いられている．例えば，個体を単位として行動規則を与えるモデルや，確率的な遷移を用いるモデル，待ち行列理論のように処理過程を抽象化する手法などが，分野に応じて活用されている．このように，現象の本質を損なわない範囲で適切に単純化し，数式や規則として表現することが重要である．

第四に，構築した数理モデルを用いて現象の振る舞いを調べる．現実の問題は解析的に解を求めることが困難な場合が多く，特に多数の要素が相互に影響し合う系では，数値計算やシミュレーションが有効な手段となる．この工程では，モデルに含まれるパラメータを変化させながら挙動を観察し，条件の違いが結果にどのような影響を与えるかを検討することで，現象の特徴や傾向を把握する．

第五に，得られた結果の解釈とモデルの妥当性の検討を行う．この工程では，シミュレーションや計算結果を単なる出力として受け取るのではなく，現象が「どの条件で」「どの程度」「どのように」発生するのかを読み解き，背後にある構造的要因を整理する．あわせて，得られた結果が観測事実や既存の知見と整合しているかを確認し，再現が不十分な点や過度に単純化された挙動については，モデルの前提条件やパラメータ設定の見直しを行うことで，改善の余地を明らかにする．

第六に，得られた知見をもとに改善策や応用可能性について検討を行う．数理モデルは現象理解のための道具にとどまらず，条件変更が結果に与える影響を事前に検討する手段としても活用できる．このように，数理モデリングは複雑な現象に対して体系的な分析を可能にし，問題解決や意思決定支援に寄与する有効な方法である．

（※文責: 坂井康大）

2.1.2　数理モデリングを用いる意義

数理モデリングを用いる意義は，単に数式やシミュレーションを形式的に適用することにあるのではなく，対象となる現象に対する理解を深め，思考の整理を支援する点にある．モデル構築の過程では，「どの要素が現象に本質的な影響を与えているか」「どの要素は簡略化してもよいか」といった選択が求められ，これらの判断がモデルの視点や表現形式に反映される．したがって，数理モデルは単なる客観的な再現物ではなく，「作成者が現象をどのように解釈しているか」を示す認知的枠組みでもあり，現実の理解を補助する解釈装置として機能する．

また，数理モデルは必ずしも一意に定まるものではなく，同じ現象に対しても目的や前提条件に応じて異なる形を取り得る．しかし，これは欠点ではなく，モデルの有効性が「現実を完全に忠実に再現できるかどうか」ではなく，「課題解決に必要な視点を適切に提供できているかどうか」によって評価されることを意味する．適切な抽象化が行われたモデルは，複雑な現象を可視化し，影響因子の整理や施策検討を容易にする一方，不適切な抽象化は現象の本質を見失わせ，分析や改善策の検討を困難にする可能性がある．そのため，数理モデリングにおいては，モデルの目的と役割を明確にした上で，現実の状況に照らして妥当な視点を選択することが重要である．

以上の一般的な特性を踏まえ，本プロジェクトでは数理モデリングという手法を採用した．本プロジェクトの対象である食堂内の混雑は，利用者の移動，待ち行列の形成，座席状況，レイアウト構造など複数の要素が同時に作用することで生じるため，直感的判断や経験則のみでは，原因や改善策の効果を十分に見極めることが難しい．そこで，現実の状況を計測や観察によって把握し，得られた情報を数学的な枠組みに整理することで，問題の構造を明確化し，混雑発生の要因や時間変動の傾向，対策導入時の影響範囲を定量的に把握することを目指した．

このように，本プロジェクトにおいて数理モデリングを用いることは，現象の理解と可視化，分析の体系化，および改善策の検討と評価を支援し，混雑緩和に向けた現実的かつ根拠ある提案を行うための有効な手段として位置付けられる．

（※文責: 坂井康大）

2.2 先行研究

2.2.1 Social Force Model

Social Force Model は，Helbing らによって提案された歩行者挙動の数理モデルであり，人間の移動を物理的な力の作用として表現する点に特徴を持つ [5]．このモデルでは，歩行者が目的地に向けて進もうとする「希望推進力」，他者や障害物との接近・衝突を避けようとする「反発力」など，複数の仮想的な力の合成によって進行方向と速度が決定されると考える．こうした枠組みによって，個人の意思決定の微視的挙動と，群集全体の流れとしての巨視的現象を同一モデル上で扱うことが可能となる．

このモデルは，駅や空港ターミナル，イベント会場などの高密度空間における歩行行動の再現に広く利用されており，特にボトルネック付近での密集や，流入密度の増大に伴う歩行速度の低下，避難行動時の群集の詰まりといった現象が再現できることが先行研究で示されている [6][7]．さらに，通路幅や障害物配置，出口位置といった空間的パラメータを変更しながら群集流動への影響を評価できるため，建築計画や避難計画の改善に向けた設計支援ツールとしても活用されている．

一方で，Social Force Model にはいくつかの課題も指摘されている．力の大きさや作用範囲，個人差を反映するパラメータの設定が結果に大きく影響するため [8]，対象とする空間や利用者属性に応じた適切なキャリブレーションが不可欠である．また，心理的判断や集団行動における戦略的選択といった人間特有の行動特性については十分に表現できず，モデル化には一定の限界が存在する．そのため，Social Force Model は万能な再現装置ではなく，観測データや現場調査を併用し，モデルの前提条件や適用範囲を明確にした上で，慎重に利用することが求められる．

以上のように，Social Force Model は，人の移動や混雑の発生メカニズムを数理的に捉えるための有力な枠組みとして，多くの研究領域で用いられており，空間内の動線設計や混雑緩和策の検討において重要な役割を果たしている．

（※文責: 坂井康大）

2.2.2 待ち行列理論

待ち行列理論は，利用者がサービスを受ける際に形成される行列の挙動を，確率論と統計学の枠組みに基づいて解析する理論である．到着率やサービス時間，窓口数などをパラメータとし，平均待ち時間，行列長，システム混雑度といった指標を定量的に評価することで，サービス提供システム全体の効率を分析できる．この理論は通信ネットワークや生産管理，交通工学，医療現場など，待ち時間や混雑が社会的課題となるさまざまな領域で応用されている．

先行研究では，最も基本的な M/M/1 型モデルから，複数窓口モデル（M/M/c 型），優先度・予約・バッチ処理などを考慮した拡張モデル，さらに時間帯ごとの到着率変動を取り入れた非定常モデルなど，多様な応用が議論されている [9]．これらの研究により，「窓口数の増減が待ち時間に与える影響」，「サービス設計の違いによる行列長の変化」，「誘導方式や案内情報の提示が利用者行動に与える心理的影響」などが理論的に整理されてきた．

一方で，待ち行列理論は数理的単純化を前提とすることから，実際の空間構造や人の移動を直接扱うことが難しいという欠点も存在する．特に，行列が通路や座席周辺を占有し，他の動線を阻害する環境においては，理論モデルと現実挙動の間に乖離が生じる場合がある．そのため，近年では

待ち行列理論を基礎としつつ，シミュレーション手法やエージェントモデルと組み合わせることで，理論的整合性と空間的リアリティを両立させる研究が進められている．

このように，待ち行列理論は混雑現象の発生要因を定量的に理解するための理論的基盤として機能し，サービス効率化や混雑緩和策の設計において依然として重要な位置づけを持つ．

（※文責: 坂井康大）

2.3 修得した知識

2.3.1 輪講会実施の意義と目的

本プロジェクトの根幹には,「数理モデリングを用いて，函館市や公立はこだて未来大学に関わる人々にとって有益な成果物を作成する」という目標が置かれており，数理モデルの構築は活動全体を支える中核的要素である．混雑という現象は，利用者行動・空間レイアウト・時間変動など複数の要因が相互に作用して生じる複雑系であり，分析には一定の理論的素養と実装技術が不可欠である．そのため，数理モデルを「使う」だけでなく，自ら「構築できる」状態を目指す必要があった．

しかし，プロジェクト開始当初において，多くのメンバーは数理モデル構築の経験が乏しく，モデル設計やパラメータ設定の根拠となる理論的理解が十分に備わっていなかった．加えて，数理モデリングを進める際には，微分方程式の離散化，収束判定，数値誤差の取り扱いなど，数値計算法の知識を要求される場面が多い．これらを理解しないまま実装段階に進むことは，モデルの妥当性評価やパラメータ調整が困難になる可能性が高く，結果として再現性や信頼性の低いモデルに至る懸念があった．

そこで，本プロジェクトでは基礎理論の学習と実装技術の獲得を目的として，既存の数理モデルや数値計算法を題材にした輪講会を実施する方針を採用した．輪講会という形式を選択した理由は，単なる個別学習ではなく，理解の相互補完・知識の共有・疑問点への即時対応が可能となる学習環境を整えるためである．たとえ最終的に直接使用しない手法であっても，選択や除外の判断には一定の知識が必要であり，学びを深めることを優先する姿勢で輪講会を進めることにした．

以上を踏まえ，輪講会の目的は次の 2 点に整理される．

・プロジェクトメンバーが数理モデルの構築経験を積むとともに，数値計算法に関する基本的な知識を身につけること．

・輪講会で得た知識をもとに，プロジェクトテーマに沿った数理モデルを作成する際，必要な知識を自律的に取捨選択できるようになること．

このように，輪講会は単なる勉強会ではなく，プロジェクトの知識基盤を整備し，最終成果の質を左右する重要な工程であった．

（※文責: 坂井康大）

2.3.2 輪講会で使用した書籍と選定理由

本プロジェクトの輪講会では，森北出版の『Python による数値計算法の基礎』[10] を主教材として採用した．同書は数値計算に必要な基礎理論を段階的に解説しながら，Python を用いた実装手法へと接続する構成を特徴とする．各手法の背景理論だけでなく，実際に手を動かしてコードを書きながら理解を深められる点が，本書を選定した理由である．

書籍には，ニュートン法・オイラー法・スプライン補間・常微分方程式の数値解法など，モデル構築に関連する技術が体系的にまとめられている．特に，本プロジェクトで採用した Social Force Model に内在する微分方程式の数値的取り扱い，および離散ステップによる時間発展の再現に応用できる内容が多く含まれており，学習と実務が密接に接続する形で活用することができた．演習問題が豊富であった点も有用であり，理解の定着と実践的スキルの習得を後押しした．

さらに，本書が Python を前提としている点も選定理由として大きい．本プロジェクトのシミュレーション実装環境が Python ベースであったため，学習内容がモデル実装や検証へ直接反映されやすい状況が整い，「学んだことを即座に試す」という理想的な学習ループを形成することができた．このように，本書は輪講会の目的に適合し，プロジェクト全体の基礎力向上に貢献した教材であったと言える．

（※文責: 坂井康大）

2.3.3　輪講会での学習内容

本プロジェクトの輪講会では，数理モデリングの基盤となる数値計算法および関連理論を体系的に学習した．目的は単なる知識習得ではなく，「モデル構築に必要な理論を理解し，自ら適切な手法を選択できる状態になること」であり，学習内容は実装と結びつけながら進められた．以下に，各回の学習内容と学習を通じて得られた知見を述べる．

第 1 回　非線形方程式と初期値依存性

初回では，非線形方程式の解法としてはさみうち法，二分法，ニュートン法を学んだ．特に，数値手法には適用条件が存在することを確認し，「f(a) と f(b) が異符号である」という前提が崩れた場合には，はさみうち法や二分法が適用できない点を実例を通じて理解した．また，ニュートン法において f’(x) が 0 に近づく場合，反復計算が発散し不安定化することを実装で確認した．

第 2 回　数値積分による離散化の基礎

第 2 回では，台形法とシンプソン法を通じて，連続量を離散的に扱う技術を学んだ．台形法は近似精度は比較的低いが実装負荷が小さく，試作段階や初期検証に適していることを確認した．一方，シンプソン法は 2 次補間を用いることで高精度な近似が可能となり，特に滑らかな関数に対して有効であることを理解した．また，「分割数が偶数でなければならない」「端点誤差が誤差伝搬を引き起こす」といった実装上の注意点や，重み係数の扱いを誤ると精度が崩れることも体験的に把握した．これらは後の時間発展シミュレーションにおける時間ステップ選択や刻み幅調整へ応用可能な重要知見であった．

第 3 回　データ近似と補間法

第 3 回では，最小二乗法・ニュートンの補間法・スプライン補間法を取り上げ，実測データや観測データを数理モデルに取り込むための技術を学習した．最小二乗法では，誤差評価指標（残差平方和）を基準として回帰直線を決定し，観測値と理論との整合性を定量的に評価する手順を理解した．ニュートンの補間法では，多項式による近似が必ずしも高次数ほど良い結果をもたらさないこと，過剰次数はランゲ現象を引き起こす可能性があることを議論した．スプライン補間においては，「3 次関数が滑らかさと計算負荷のバランス点になる」という知見が得られ，グラフ描画により滑らかさの違いを視覚的に確認した．これらは，食堂内の人流データ補間や混雑変動曲線の滑らかな表現に応用可能であると考えられる．

第 4 回　線形方程式と反復解法

第 4 回では，緩和法・掃き出し法・ガウス・ザイデル法を学び，「直接法と反復法の違い」「収束条件」「計算負荷と精度のトレードオフ」を整理した．特にガウス・ザイデル法において，対角優位行列であることが収束性を保証する条件となる点を学び，解の安定性が問題構造に依存することを理解した．Python による反復計算の実行を通じ，収束の速度と近似解の精度の関係を体験的に把握し，パラメータ調整の重要性を共有した．この学びは，後のシミュレーションの行列計算部分に直接応用され，計算コストと精度バランスの判断基準を形成する基礎となった．

第 5 回　常微分方程式の数値解法

第 5 回では，常微分方程式の数値解法について学習し，テイラー法，オイラー法，ルンゲ＝クッタ法といった代表的な手法を通して，連続的に変化する現象を離散的な計算によって近似的に扱う考え方を理解した．現実世界における多くの現象は時間とともに連続的に変化するが，計算機上ではそれをそのまま扱うことができないため，時間を刻み幅ごとに区切り，数値的に解を追跡する必要がある．本講義では，その基本的な枠組みと，各手法の特徴や限界について体系的に学ぶことができた．

まずテイラー法では，関数を微分係数によって局所的に近似する考え方を通して，数値解法の理論的背景を理解した．解析解が求められない場合でも，現在の状態とその変化率を用いることで次の状態を予測できる点は，数値計算全般に共通する重要な発想であると感じた．その後に扱ったオイラー法は，テイラー展開の一次近似に基づく最も基本的な数値解法であり，計算手順が単純で実装が容易であるという利点を持つ．一方で，刻み幅の設定によって誤差が大きく左右され，刻み幅を大きくすると解が不安定になったり，現実とは異なる挙動を示す可能性があることを確認した．

これに対して，ルンゲ＝クッタ法は，1 ステップ内で複数回傾きを評価することで，より高い精度を実現する手法である．計算量は増加するものの，刻み幅に対する依存性が比較的小さく，精度と安定性のバランスに優れている点が特徴的である．特に，オイラー法と比較することで，数値解法において「計算コスト」と「解の信頼性」をどのようにトレードオフとして考えるべきかを具体的に理解することができた．また，高階の常微分方程式を一次の連立常微分方程式へ変換することで，これらの数値解法を適用可能にするという考え方も重要であり，モデル化の段階での工夫が計算結果に大きな影響を与えることを学んだ．

さらに，Python を用いた数値計算の実装を通して，人流モデルや物理現象の時間発展を実際にシミュレーションしたことで，数式として与えられた数理モデルが，時間の経過とともにどのような挙動を示すのかを視覚的かつ直感的に理解できた．単に理論を学ぶだけでなく，実装と結果の解釈を行うことで，数理モデルと現実現象との対応関係をより深く考察する機会となった点が，第 5 回の講義の大きな意義であると感じた．第 6 回　偏微分方程式と空間的広がり

最終回では，差分法，FDTD 法，有限要素法（FEM）を扱い，空間的広がりを持つ現象を数値的に扱うための基礎理論を学習した．差分法では，空間・時間格子の設定が誤差伝搬に大きく影響する点を確認し，FDTD 法では波動現象の離散化過程を通じ，電磁気学的観点（アンペールの法則，ファラデーの法則）との関連も整理した．有限要素法に関しては，領域を要素に分割して解を近似する思想，ガラーキン法による誤差の直交性の概念，コロケーション法や部分領域法など手法選択の指針について理解した．これらは，食堂空間における動線干渉や領域密度変化を扱う将来的モデル拡張の選択肢となり得る知見であった．輪講会を通じて，理論学習と実装演習を反復しながら，次の 3 点を中心とした成果が得られた．

・手法を知識として「理解する」段階から，目的に応じて「選択できる」段階への成長．
・モデル構築時に必要となる前提条件・適用範囲・誤差要因の判断基準の獲得．
・Python 実装を通じた数理モデルとシミュレーションの接続的理解の定着．

これらの知見は，今後の食堂混雑モデルの開発・検証・改善における技術的基盤となり，最終的な成果物の信頼性向上に寄与することが期待される．

（※文責: 坂井康大）

第 3 章　アプローチ

本プロジェクトは，「食堂における混雑の発生要因を人流モデルによって明らかにし，混雑緩和のための具体的な改善案を提案すること」を主目的として進めた．この目的を達成するために，単にシミュレーションを構築するだけでなく，現状把握，モデル設計，検証，提案，発信という一連のプロセスを計画的に実行することを重視した．本章では，プロジェクト全体の流れと体制について述べる．

（※文責: 菊地皓太）

3.1　プロジェクトの対象範囲

本プロジェクトの対象は，公立はこだて未来大学内の食堂である．対象とする時間帯は主に昼食時のピーク帯とし，食堂営業時間である 11 時 30 分から 13 時に観測と分析を行った．対象とする現象は，「利用者の入場から退場までの一連の動きと利用者数」「待ち行列の形成・伸長・解消の過程」「ウォーターサーバーや小皿棚の利用率」「特定の場所での滞留や衝突の発生」などである．これらを数理モデルとシミュレーションによって表現し，混雑を引き起こす要因とその構造的特徴を明らかにすることを目指した．

また，本プロジェクトでは「現実の全てを精密に再現する」ことではなく，「混雑緩和策の検討に必要な要素を抽出し，再現可能なレベルでモデル化する」ことを目的とした．このため，メニューの詳細や個人の特性といった要素はモデルから切り捨て，動線，待ち行列，座席利用状況など混雑に直結する要素を優先して取り扱う方針とした．

（※文責: 菊地皓太）

3.2　プロジェクト全体の流れ

本プロジェクトは，2.1.1 で述べた数理モデリングの手順に沿って進められた．

前期の初期段階においては，連絡手段や使用するプログラミング言語・開発環境の確認を行い，輪講形式による基礎的な理論学習を通して，プロジェクトの進め方に関する共通理解を形成した．その後，学内の課題を幅広く洗い出し，検討の結果，本プロジェクトの研究対象として「食堂の混雑」を選定した．これに並行して，人流モデルやエージェントベースシミュレーションに関する基礎知識を共有し，どのようなモデルによって現象を表現できるかを検討した．

次に，食堂の現状把握を目的として現地観測を実施し，曜日・時間帯ごとの利用者数，待ち時間，混雑の状態，また実際の食堂のレイアウトなどのデータを収集した．併せて，利用者および従業員へのアンケート調査とインタビュー調査を実施し，混雑に対する主観的な負担感や現場側の課題認識を把握した．これらの結果に基づき，混雑の発生要因に関する仮説を立案し，検証すべき要素を整理した．

その後，実際の食堂レイアウトと動線を反映した二次元シミュレータの設計・実装に取り組んだ．エージェントの行動ルール（入場，注文，着席，返却，退場など）を定義し，実時間とシミュ

レーション内での仮想時間の整合性，エージェントの移動速度・大きさ，待ち行列の挙動などを調整しながら，現実の状況に近づくよう改良を重ねた．さらに，複数のレイアウト案や動線案を実装し，比較実験が可能な環境を構築した．

検証段階では，混雑を定量的に評価する指標として「人口密度」と「衝突回数」を設定し，特定時間帯におけるシミュレーション結果を比較した．行列の配置変更や通路幅の調整などの改善案を適用し，それらが指標に与える影響を観測することで，混雑緩和効果の評価を行った．

最後に，得られた結果をもとに，食堂運営側に対して実行可能性のある改善案として提案内容を整理し，スライド・ポスター・報告書として取りまとめた．また，外部向けの説明資料作成や発表練習を行い，内容の妥当性と説明の明瞭性を高めた上で，プロジェクト成果として発信した．

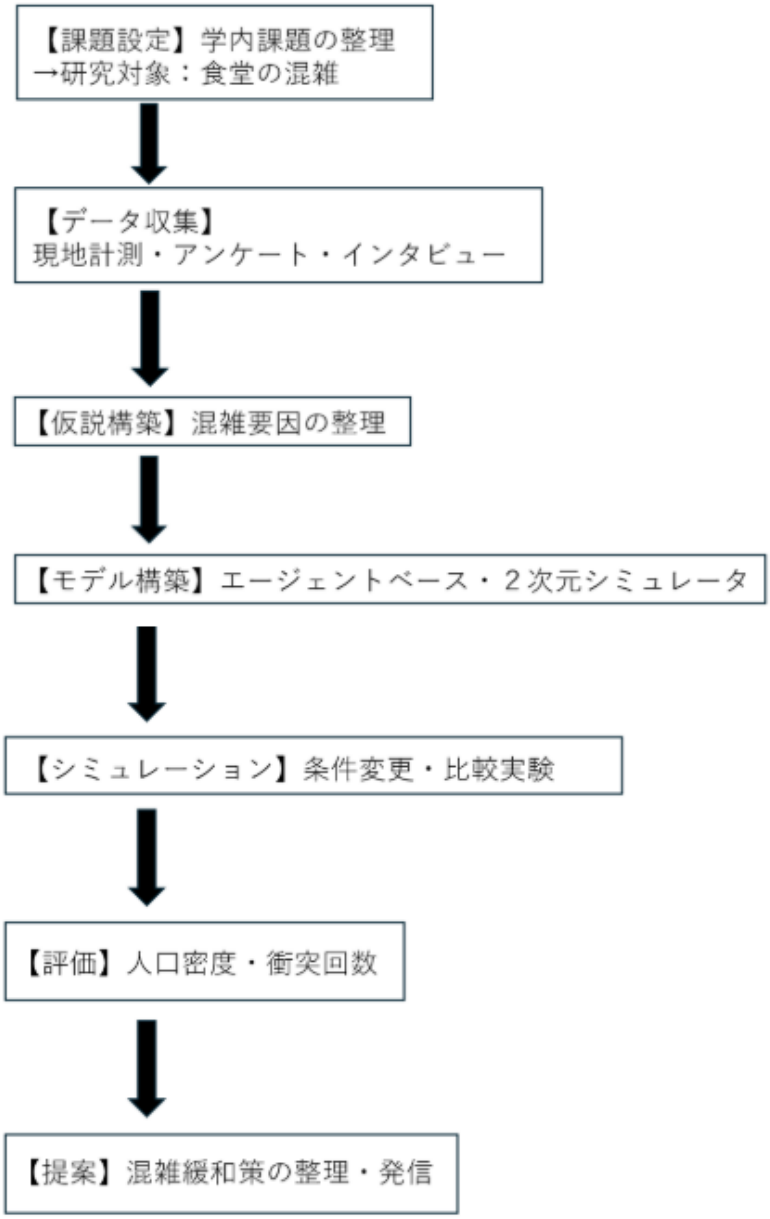

図 3.1　プロジェクト全体のフローチャート

（※文責: 西尾柊太）

3.3　コミュニケーション手段と開発環境

プロジェクトを継続的に進めるためには，技術的な準備に加えて，日常的なコミュニケーションと情報共有の仕組みが重要である．本プロジェクトでは，主に以下の環境を整備した．

・連絡手段として，対面での話し合いとチャットツールである Discord を併用し，日々の進捗報告や相談を行える場を確保した．

・Python の開発環境は，仮想環境を用いてライブラリ構成を統一し，シミュレーション実行環境の再現性を確保した．

また，ソースコード管理については Git を使用せずに各メンバーが担当部分のコードを個別に記述し，最終的な統合作業は人力でマージする手順を採用した．そのため，直接コードを照合しながら調整を行い，挙動確認を繰り返しつつ整合性を確保した．この方法は効率的とは言い難いが，コード全体の理解度を高める点や，実装意図を共有する機会を増やす点において一定の意義があった．

（※文責: 菊地皓太）

3.4　現状把握とデータ収集の方針

モデル構築の前提として，現実の状況をどの程度の精度で再現するかを定める必要がある．本プロジェクトでは，以下の観点から現状把握を行った．

定量的データの収集

・時間帯ごとの利用者数，配膳口周辺の行列長，座席の使用状況，入場から退場までの滞在時間などを実測し，人流の基本的な傾向を把握した．

定性的情報の収集

・利用者アンケートを通じて，「不便に感じる点は何か」といった主観的な意見を収集した．また，従業員へのインタビューにより，運営側から見た課題や混雑事例，改善の試みなども把握した．

これらの情報は，「どこで」「いつ」「どのような理由で」混雑が発生しやすいのかについて仮説を立てる支援となり，後のモデル設計に反映させた．

（※文責: 菊地皓太）

3.5　モデル構築とシミュレーションの方針

データ収集の結果を踏まえ，食堂内の人流を再現するモデルとしてエージェントベースモデルを採用した．エージェントベースモデルでは，個々の利用者を独立した主体として扱うことが可能である．さらに，Social Force Model を用いることで，各利用者の目的地志向の移動および他者との衝突回避行動を表現する．

シミュレーションの設計にあたっては，以下の点を重視した．

・実際の食堂レイアウトを二次元平面上に再現し，壁・通路・配膳口・返却口・座席エリアなどを明示的にモデル化すること．

・入場，注文，着席，返却，退場といった利用者の行動フェーズを定義し，各フェーズ間の遷移条件を明確にすること．

・行列の形成位置と伸長方向をモデル内で表現し，その結果として動線がどのように圧迫され，通行可能な幅がどの程度制限されるかを観察できるようにすること．

また，実時間との対応を意識し，シミュレーション上の 1 ステップと現実時間との関係を調整することで，観測データとの差異を評価できるようにした．

（※文責: 菊地皓太）

3.6　評価指標と検証の方針

構築したモデルや改善案の妥当性を評価するために，本プロジェクトでは主に次の 2 つの指標を用いた．

1 つ目：人口密度

食堂内の各領域におけるエージェント数を単位面積あたりで評価し，局所的な密集の度合いを測定した．特に出入り口付近や配膳口周辺，通路のボトルネックとなる箇所の密度を重点的に観察した．

2 つ目：衝突回数

エージェント同士が一定距離以内に接近した回数をカウントし，動線の交錯や無理なすり抜けが多発している箇所を特定した．

これらの指標を，現状レイアウトと改善案レイアウトの間で比較することで，「どの案が混雑緩和により効果的か」を定量的に判断した．あわせて，観測データとシミュレーション結果を対応させることで，モデルの妥当性についても検証した．

（※文責: 菊地皓太）

3.7　成果物の整理と発信

最終段階では，得られた知見を食堂運営側にも共有できるよう，以下の成果物の形を整えた．

・本プロジェクトの活動をまとめた最終報告書
・概要と主な結果を提示する展示用資料となるメインポスター
・本プロジェクトのアピール文

また，最終発表に向けて，限られた時間内で要点を伝えるための発表練習を行い，説明の順序や図表の構成を調整した．これにより，プロジェクト成果を学内外へ発信し，実際の混雑緩和策の検討に資する情報として提供できる形にまとめた．

（※文責: 菊地皓太）

第 4 章　現地調査

本調査では, 本学の食堂における現状の把握として，レイアウト測量・人流計測・観察といった定量的な調査と，利用者アンケート・従業員インタビューといった定性的な調査を行った．本章では，それらの調査の目的や方法について述べる．

（※文責: 菊地皓太）

4.1　レイアウト測量

シミュレータ作成の前段階として，現実の食堂空間を仮想空間に平面図として再現するためのレイアウト測量を実施した．

（※文責: 菊地皓太）

4.1.1　目的

シミュレーションでは，通路幅や障害物等の配置が，行列の形成位置や滞留の発生箇所に大きく影響する．そのため，現実の食堂をモデル化することを目的に，以下の項目を定量的に取得した．
1：空間概形と固定障害物
・壁面，柱の位置，寸法
・出入口の位置
2：サービス施設
・レジ，ウォーターサーバー，食器返却口，ゴミ箱，テラス出入口の位置
・食事提供口，小皿提供棚，食器棚の位置と寸法
3：可動物体
・テーブルと椅子の数，寸法，配置，占有面積
・テーブル周囲の通路幅

（※文責: 菊地皓太）

4.1.2　方法

本測量は 2025 年 5 月 26 日に行った．測量にはメジャーを用い，3 人で分担して行った．空間内で基準となる壁面を基準線として，主要物体までの距離を測定し，記録を行った．

まず，事前の目視確認により食堂内のレイアウトを把握し，壁，柱，テーブル，オブジェクト等の配置を紙面上に概略図として描いた．次に，2 名がメジャーを用いて壁面から各オブジェクト，障害物の端点までの距離や通路幅を測定し，残り 1 名がその数値を概略図へ記録した．完成したレイアウトの図を以下に示す．

（※文責: 菊地皓太）

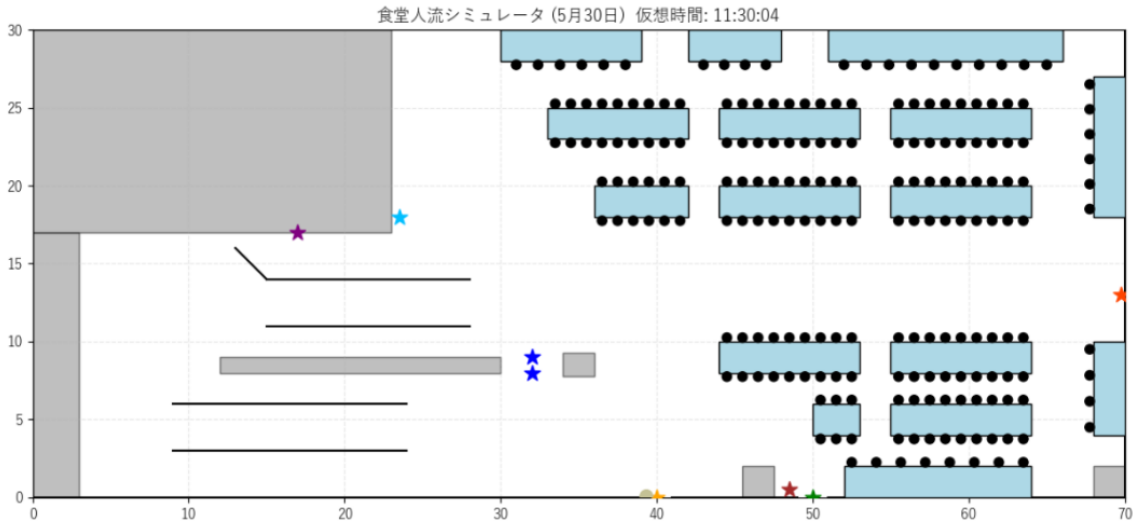

図 4.1　食堂全体図

4.2　人流計測・観察

本調査は，人流の増減傾向や行動特性を定量的に取得し，モデル構築およびシミュレーション検証の基礎データとする．

（※文責: 西尾柊太）

4.2.1　目的

本調査の目的は，食堂利用者の入退場状況および食堂内の行動特性を把握し，混雑がどの時間帯・どの場所で発生しやすいのかを明らかにすることである．特に以下の観点を重視した．

・利用者数の時間変化を定量的に把握すること

来店ピーク時間帯の特定と，入退場数の差異による滞留・混雑の発生傾向を把握する．

・利用者行動と混雑要因の関連を確認すること

支払い方法やウォーターサーバー利用，返却動線など，行動が滞留を誘発している可能性を持つ要素を観察する．

・数理モデル化を行う上で考慮すべき条件を抽出すること

入口位置，列形成位置，歩行速度の傾向，平均滞在時間など，モデル設計に反映すべき要素を洗い出す．

本調査は，単なる人数の把握に留まらず，「混雑が発生しうる構造的要因の抽出」を目的として実施した点に特徴がある．

（※文責: 西尾柊太）

4.2.2　方法

計測は，食堂の開放時間である 11 時 30 分から提供終了時間である 13 時 00 分までの間に，1 分毎の入場者数及び退場者数をカウントする形式で実施した．計測手順を以下に示す．

1：基本計測項目

・入場者数（1 分毎）

・退場者数（1 分毎）

2：行動観察項目

・計測と並行し，次の点を観察した．

・支払い準備状況：生協アプリを事前起動しているか

・支払い方法の違い：現金利用かアプリ決済か

・ウォーターサーバー利用人数：動線滞留の要因把握

・トレーの有無：退場理由（食事終了／外部持ち出し）の推定

計測は原則として手や頭の中で人数をカウントし，各曜日の結果を Excel で週毎に集計した．特に混雑が激しい時間帯においては，入退場が同時に発生するケースが多く，片方を頭でカウントし，もう一方をスマートフォンのアプリケーションを用いて計測するなど，状況に合わせて計測を行った．

調査期間

・期間：2025 年 5 月 30 日（金）～2025 年 7 月 1 日（火）

・頻度：可能な限り毎授業日の昼時間帯に実施

なお，13 時 00 分の時点では食堂内に利用者が残っていることが多く，入場者数と退場者数にズレが生じる場合があることを考慮し，可能な日は 13 時 00 分以降も引き続き計測を行った．

本調査は，人力での計測による誤差発生の可能性を認識したうえで，「現場での行動傾向を掴むための一次情報の取得」を優先したものである．数理モデル構築にあたっては，計測値を基準としつつ，合理的な範囲で抽象化・補正を行った．

（※文責: 西尾柊太）

4.3　利用者アンケート

本アンケート調査は，混雑に対する利用者の主観的負担を把握し，「どの場面でストレスが最も大きく感じられているか」を明確にする．

（※文責: 西尾柊太）

4.3.1　目的

本アンケートの目的は，混雑が利用者に与える心理的影響を把握し，現地観測では把握しにくい「不便さの実感」を数値として取得することである．すなわち，本アンケートは，数値計測に対し主観評価を重ね合わせることで，課題把握を定性的に行うための調査である．

（※文責: 西尾柊太）

4.3.2　方法

対象：学内の学生 100 名
形式：シール貼付式／単一選択式
場所：食堂前通路（通行者の動線を阻害しない位置）
　回答者には以下の 4 項目から最もストレス要因に該当するものを 1 つ選択してもらった.
・食事提供の行列・待ち時間
・レジの行列・待ち時間
・着席場所の確保
・食器の返却待ち
　シール方式を採用した理由は次の通りである.
・停滞時間が短く協力を得やすい
・回答ハードルが低く，母数を確保しやすい
・リアルタイムで回答が可視化され，誘因効果が生まれる
　これにより，短時間ながら多くの学生から回答が得られ，混雑に対する負担感の傾向を把握することができた.

（※文責: 西尾柊太）

4.4　従業員インタビュー

　本インタビュー調査は，現場で運用に携わる従業員の視点から混雑の原因や改善可能性を把握することを目的として実施した.

（※文責: 西尾柊太）

4.4.1　目的

　本インタビュ－の目的は「利用者側からは見えにくい運用上の課題を把握すること」「混雑要因の構造分析に必要な“運営側の視点”を補完すること」「シミュレーション改善案が現実的に実行可能かを評価するための材料を得ること」である．これにより，観測・アンケートでは得られない「現場感覚に基づく課題」が抽出され，改善案検討の現実性を高める根拠となった.

（※文責: 西尾柊太）

4.4.2　方法

実施日：2025 年 10 月 31 日（金）
場所：学内食堂
形式：事前質問票＋自由回答形式の半構造化インタビュー
記録方法：メモ記録後に内容整理・カテゴリー分類
　質問内容は以下の 5 項目を中心に設定し，回答の自由度を確保する形式とした.

・営業時に「ここがスムーズになれば助かる」と感じる点
・混雑が発生する主な原因の認識
・シミュレーション上で検証してみたい改善案
・学生利用者への要望
・日常業務で最も負担が大きい場面について

自由回答形式とした理由は，選択肢式では拾いきれない現場特有の意見を得るためであり，表面的な課題だけでなく背景にある構造的要因（作業動線，機器性能，人的負荷など）を抽出することを狙いとした.

（※文責: 西尾柊太）

第 5 章　現地調査結果と仮説

本章では，計測・観察，利用者アンケート，および従業員インタビューについての分析と，そこから得られる仮説について述べる．

（※文責: 西尾柊太）

5.1　収集データ分析

5.1.1　人流計測・観察から得られた知見

まず，計測および観察の結果から，本学食堂には特定の時間帯に混雑が顕著に高まる「ピーク」が存在することが確認された．特に混雑が目立ったのは，食堂開放直後に相当する 11 時 30 分前後と，2 限終了時刻に相当する 12 時 10 分前後の二つの時間帯である．これらの時間帯には，授業の区切りや休憩時間と重なることから，多くの利用者が短時間に集中して食堂を利用しようとする傾向が見られた．6 月 23 日から 6 月 27 日までの期間に 1 分ごとの入場者数を計測し，それを示した図 5.1 では，曜日ごとの違いはあるものの，いずれの日においても同様の時間帯で入場者数が急増している様子が確認できる．この結果は，混雑が偶発的なものではなく，時間割構造や昼食行動の規則性と密接に結びついていることを示唆している．

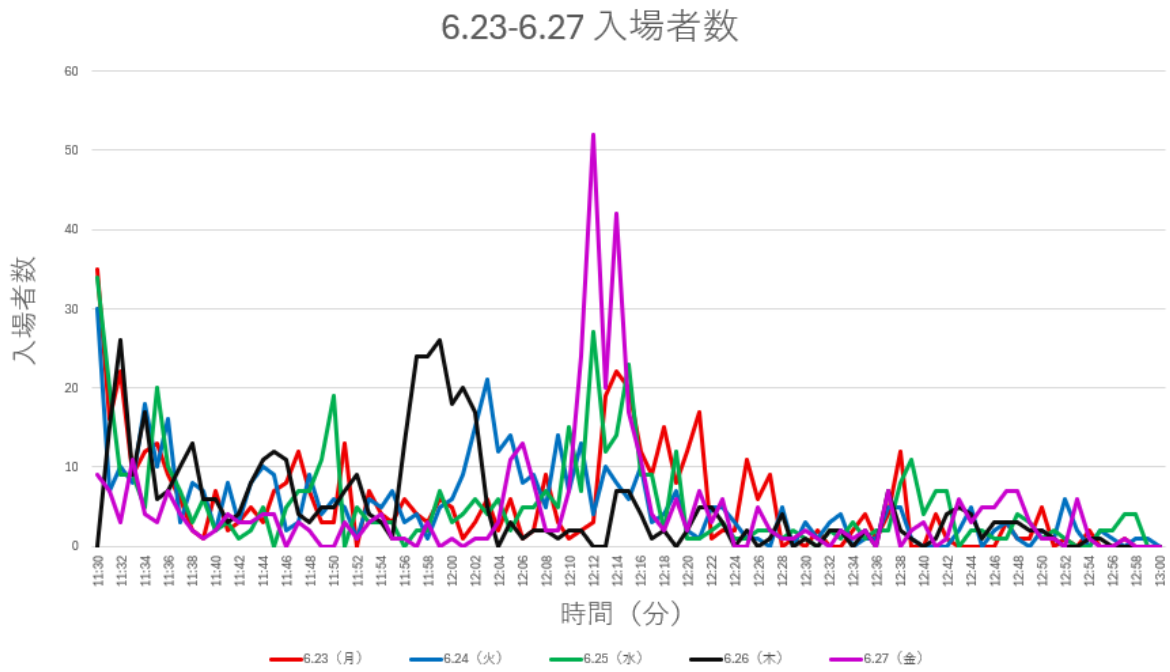

図 5.1　入場者数グラフ

赤色が月曜日，青色が火曜日，緑色が水曜日，黒色が木曜日，紫色が金曜日を表している．

次に，混雑がピークに達した際の行列形成と動線への影響について整理する．最も混雑する時間帯には，食堂内で非常に長い待ち行列が形成される．特に昼休みなどのピーク時には，行列が食堂

内のスペースだけでは収まりきらず，出入口付近から壁際まで伸び，さらに折り返すような形で二重三重の列が生じる様子が観察された．このような折り返し行列は，混雑状況が極めて深刻な場合にのみ現れる現象であり，通路の占有と動線の遮断を通じて，通行の妨げとなっている．

（※文責: 西尾柊太）

5.1.1.1　行列による動線遮断

図 5.2 に示したように，行列の位置によっては出入口付近をほぼ完全にふさいでしまうことがあり，食堂に入ろうとする利用者と退場しようとする利用者の動線が交錯し，互いに行き違うことが困難になる場面が多く見られた．特に，入場と退場が重なりやすい 12 時 20 分前後には，出入口周辺で利用者が立ち止まったり，進行方向を変えたりする様子が頻繁に観察され，そのことがさらなる滞留や混乱を招いていると考えられる．利用者は列の最後尾を探すために一度足を止めて周囲を見渡す必要があり，この「列の位置を探す」という行動自体が通路上での一時的な閉塞を引き起こし，結果として混雑を増幅する要因となっている．

さらに，図 5.3 に示したように，行列が壁際まで到達した際には，壁際の座席を利用している人のすぐ背後に待ち行列が形成される状況も確認された．このとき，テーブルと列との間の通路幅は十分に確保されておらず，座席に座っている利用者と列に並んでいる利用者が非常に近い距離で存在するため，空間的に強い圧迫感を生じさせている．また，図 5.4 に示すように，座席利用者が食事を終えて返却口に向かおうと立ち上がる際には，背後に並んでいる利用者にぶつからないよう注意を払いながら慎重に移動する様子が観察され，その動きがスムーズな移動を阻害していることも示唆された．

以上のことから，待ち行列の形成は単に通路を物理的にふさいでいるだけではなく，座席利用者の移動の自由度を低下させ，食堂全体の利用快適性を大きく損なっていることが分かる．行列の存在によって，人が「座る」「立つ」「歩く」といった基本的な行動にも制約がかかり，空間そのものが動きづらい状態へと変質しているといえる．このような状況は，動線設計の見直しやレイアウトの改善によって緩和できる可能性が高く，構造的な改善策の検討が強く求められる．

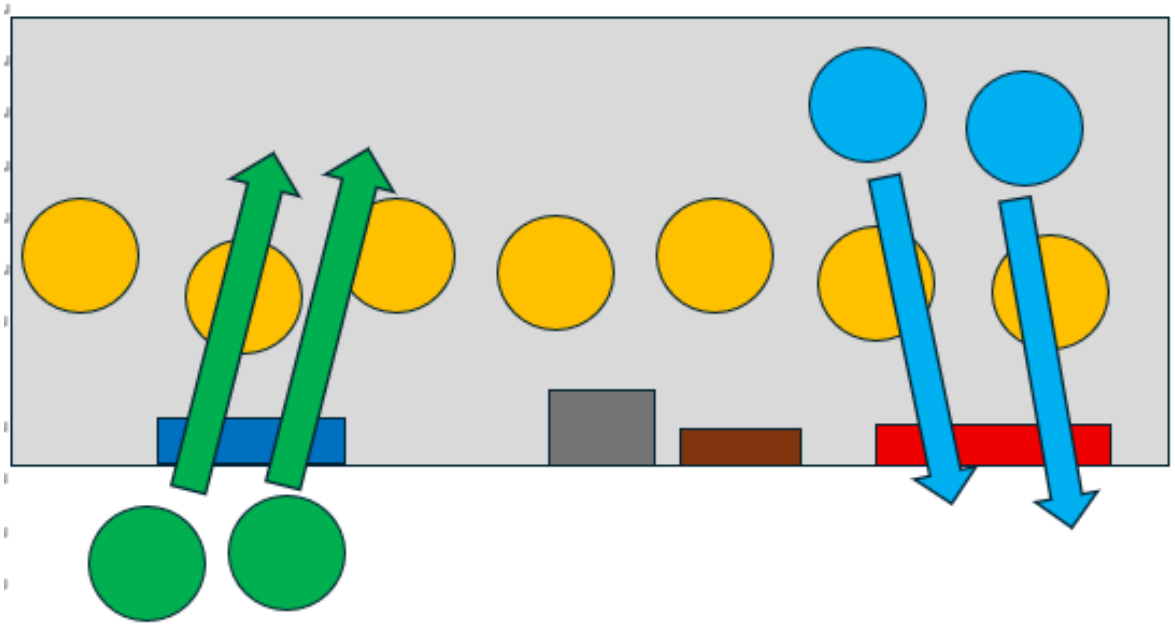

図 5.2　入口・出口付近の模式図

食堂の入口・出口付近の図
・緑色の丸：入場しようとする人
・水色の丸：退場しようとする人
・オレンジ色の丸：食堂で並んでいる人
・青色の長方形：入口
・赤色の長方形：出口

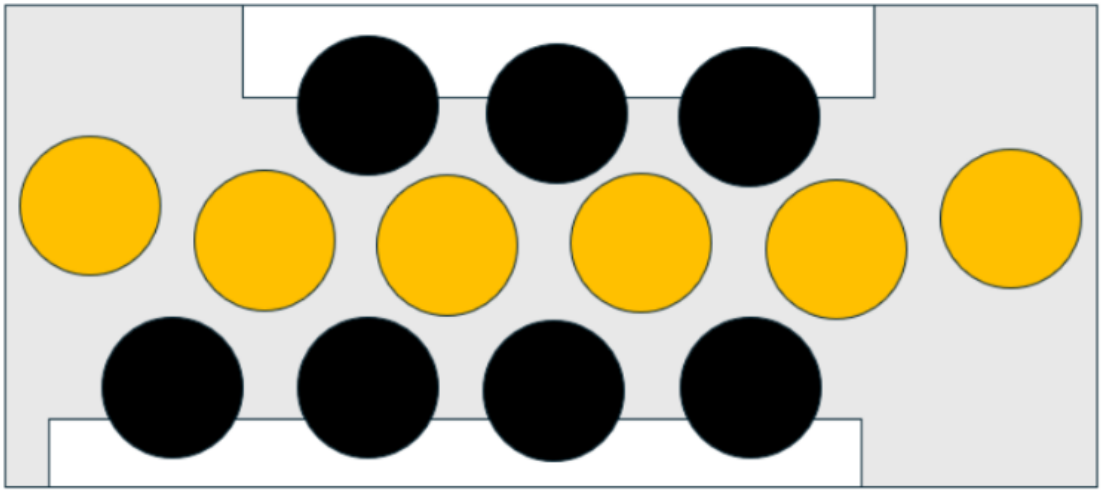

図 5.3　壁際まで行列が到達した際の模式図

・黒色の丸：座席に座っている人
・オレンジ色の丸：食堂で並んでいる人
・白色の長方形：テーブル

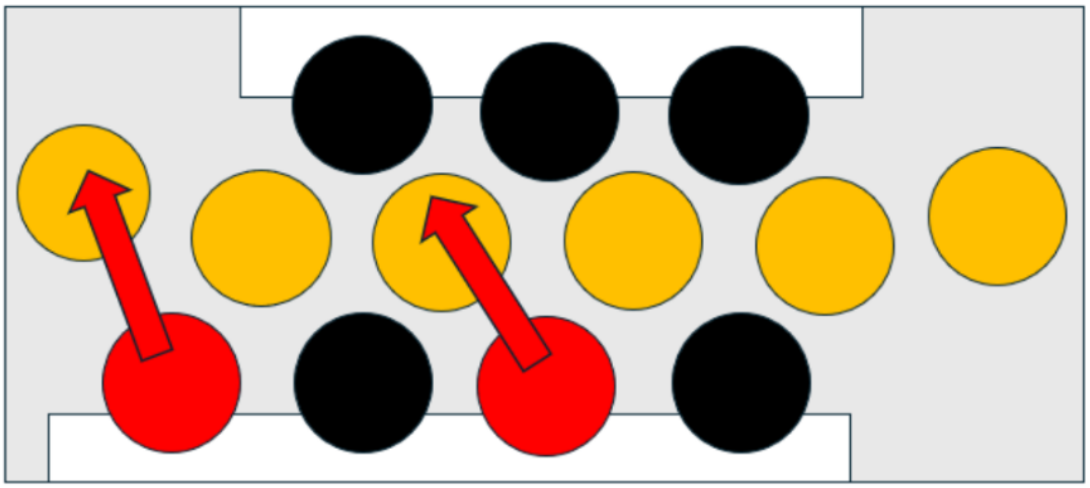

図 5.4　返却口へ向かう座席利用者の動線と行列との干渉を表している図

・黒色の丸：座席に座っている人
・オレンジ色の丸：食堂で並んでいる人
・赤色の丸及び矢印：返却口に向かおうとしている人

・白色の長方形：テーブル

（※文責: 西尾柊太）

5.1.1.2　出入口の逆利用

また，計測・観察を通して，入口と出口の「逆利用」が少なからず発生していることも分かった．つまり，本来は入口から入場すべきところを出口側から入ってしまう利用者や，逆に入口側から退場する利用者が一定数見られた．これらの逆方向の移動は，混雑時には列を避けるための迂回行動として行われている側面もあると考えられるが，比較的空いている時間帯にも発生していたことから，単に混雑回避だけが理由ではないことが示唆される．約 1 か月にわたる観察の中で，入口から入ろうとする利用者と出口側に向かって退場しようとする利用者が狭い通路で正面から向き合い，衝突しそうになる場面が複数回確認された．これらの状況は，動線ルールの明示不足や，利用者側のルール認識の曖昧さが一因となっている可能性が高いと考えられる．

（※文責: 西尾柊太）

5.1.2　利用者アンケート結果

食堂利用者を対象としたアンケート結果について述べる．アンケートでは，「食堂利用においてどの場面を最もストレスに感じるか」という質問に対し，あらかじめ用意した「食事提供の行列・待ち時間」「レジの行列・待ち時間」「着席場所の確保」「食器の返却待ち」の 4 項目から 1 つを選択してもらった．その結果を集計したものが図 5.3 である．

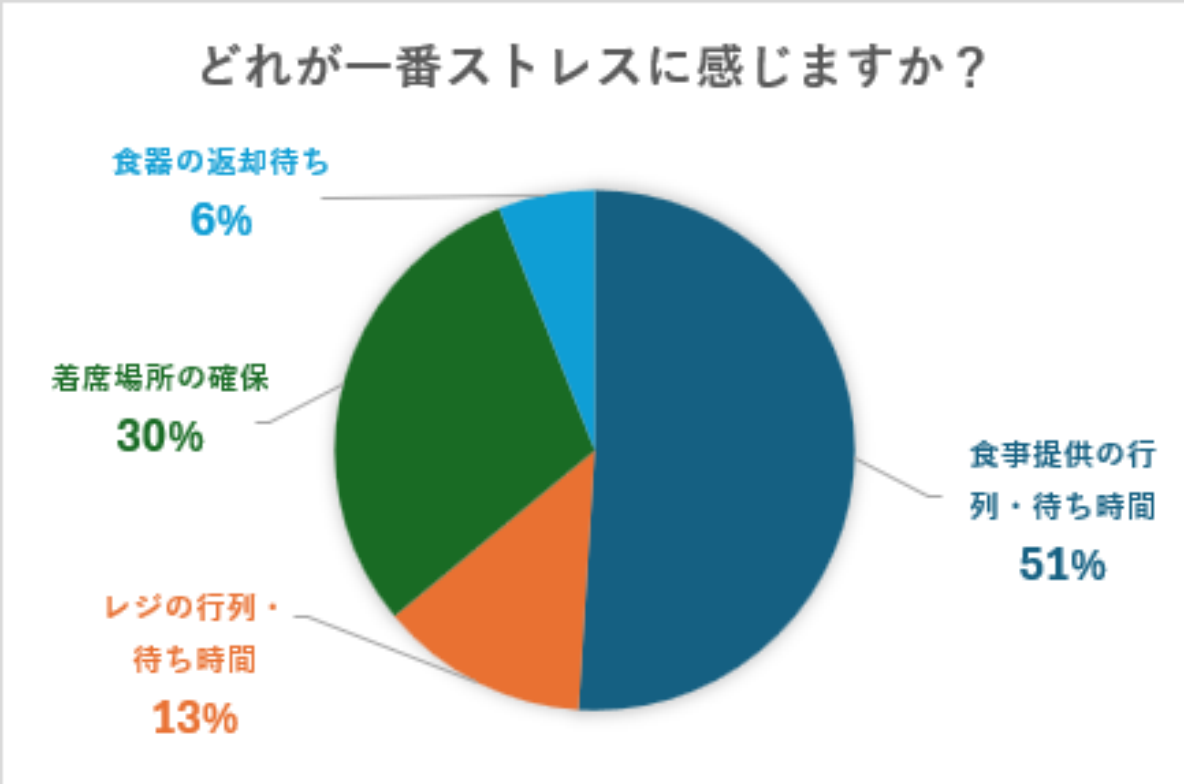

図 5.5　利用者アンケート結果

図 5.3 から明らかなように，「食事提供の行列・待ち時間」を最も負担に感じると回答した学生

が 51 名と，全体の中で最も多い結果となった．次に多かったのが「着席場所の確保」で 30 名，続いて「レジの行列・待ち時間」が 13 名，「食器の返却待ち」が 6 名という順であった．この分布から，多くの学生にとって「食事を受け取るまでの過程」と「座席を確保するまでの過程」が，食堂利用における主要なストレス源になっていることが分かる．

特に，「食事提供の行列・待ち時間」と「レジの行列・待ち時間」という，いわゆる「並んで待つ時間」に関する二つの項目だけで合計 64 名分の回答があり，全回答の過半数を占めている．これは，食堂の混雑が単に物理的な人口密度の高さとして現れているだけでなく，「自分の順番が来るまで動けない」「列が進まないことに対するストレス」といった心理的負担としても強く認識されていることを示している．すなわち，混雑の問題は「人が多いこと」そのものよりも，「待ち時間が長く感じられること」「自分の時間が制約されること」に起因する側面が大きいと解釈できる．

この結果は，本プロジェクトにおいて混雑緩和策を検討する際に，「待ち時間の短縮」や「列が停滞しないようにする工夫」が重要な観点であることを裏付けている．同時に，座席確保に関する回答数も多かったことから，「座れるかどうか分からない」という不確実性も，利用者のストレス要因として無視できないことが分かる．

（※文責: 西尾柊太）

5.1.3　従業員インタビュー結果

従業員インタビューからは，利用者側の視点だけでは見えにくい運営上の課題や，作業現場での実感が多く得られた．まず，「営業時に『ここがスムーズになれば助かる』と思う点」に関する質問では，「『どうぞ』と声をかける前にあらかじめ注文内容を決めておいてほしい」という意見が挙げられた．提供カウンター前では，従業員側としては常に 2 人程度の利用者が待機しており，一人が受け取りを終えたらすぐに次の一人が注文に進むという一定のリズムが維持される状態が理想であると説明された．しかし実際には，注文の順番が来てからメニューを選び始める利用者も多く，その分だけ提供プロセスが遅延し，全体の行列進行が滞りやすくなるという指摘があった．

また，スマートフォンの生協アプリに関する要望も示された．アプリは一定時間が経過するとセッションが切れて再認証が必要になる仕様となっており，決済直前になって画面更新が必要となる場合がある．このとき，利用者がその場でアプリを立ち上げ直したり，残高をチャージしたりすることで，会計に時間がかかり，その分レジ前での待ち時間が延びてしまうとのことであった．

「混雑が発生する主な要因」に関する質問に対しては，生協マネーの残高不足により，会計時にその場でチャージを行う利用者が一定数いることが挙げられた．これは，レジの処理リズムを乱す要因となり，行列全体の進行を遅らせることにつながると指摘された．さらに，12 時 40 分に到着するバスの利用者が一斉に来店することや，2 限終了直後である 12 時 10 分前後に多くの学生が到着することなど，時間帯による利用者の集中も混雑の大きな要因として示された．これらの時間的要因は，短時間に利用者が集中するという構造的な問題であり，利用時間の分散ができれば業務負担が軽減される可能性があるとの意見があった．

「コンピュータ上で人の流れをシミュレーションできる場合に検証してみたい改善策」について尋ねたところ，調理スタッフの人数配置に関する提案が挙げられた．特に，カレーや丼ものを提供するカウンターにスタッフを一人追加することで，盛り付け作業と提供作業を分担し，提供速度を向上させられる可能性があるとのことであった．また，レジの台数を増やすことや，食券システムを導入することによって会計処理の効率化を図るといった案も，シミュレーション上で検証してみ

たい施策として挙げられた．

「学生への要求」としては，「『どうぞ』と言われるまでに注文内容を決めておいてほしいこと」や，「生協マネーのチャージおよびスマートフォン画面の準備を事前に済ませてほしいこと」が指摘された．また，食器返却時に洗い場へ食器を入れる際，水が跳ねて周囲に飛散しないよう配慮してほしいという要望もあり，作業環境の安全性や衛生面への配慮が求められていることが分かった．

最後に，「働いていて最も大変だと感じること」としては，メニューによって盛り付け作業に時間がかかる点や，重い食材や物品を運ぶ作業が身体的負担になっていることが挙げられた．さらに，曜日ごとの利用者数の変動が事前にある程度予測できれば，仕込み量の調整や廃棄ロスの削減に活用できるという意見も得られた．この点は，将来的に来客予測と連動した運営最適化の可能性につながる重要な示唆であると言える．

（※文責: 西尾柊太）

5.2　結果に基づく仮説

5.2.1　待ち行列の観点

ここでは，収集したデータに基づき，待ち行列の観点から立てた仮説について述べる．計測および観察の結果から，混雑のピークである 11 時 30 分前後および 12 時 10 分前後には，待ち行列が入口や出口付近まで長く伸びてしまう状況が頻繁に確認された．このとき，食堂に入場しようとする利用者と，食事を終えて退場しようとする利用者の動線が，待ち行列と交錯する様子が繰り返し観察されている．

このような状況は，待ち行列が一列で形成されていることにより，奥行き方向に伸びやすく，結果として出入口付近のスペースを大きく占有してしまうことが一因であると考えられる．列の長さが増すことで，通路の有効幅が狭まり，入退場時のすれ違いが難しくなり，立ち止まりや迂回行動が増加し，さらなる混雑を招いていると推察される．

以上の観察結果を踏まえ，本プロジェクトでは「待ち行列を 2 列にすることで，入口・出口付近の動線を確保できるのではないか」という仮説を立てた．具体的には，列を 2 列に分割することで行列の奥行きを短くし，出入口付近の通路に一定の余白を生み出すことができれば，人の流れが停滞しにくくなり，入場者と退場者が互いに進路を譲りながらスムーズに動けるようになるのではないかと考えた．この仮説は，単に列の本数を増やすという発想ではなく，「どの方向に行列を伸ばすか」「どの領域に混雑を集約するか」という空間配置の観点に基づくものである．

（※文責: 西尾柊太）

5.2.2　レイアウトの観点

次に，レイアウトの観点から立てた仮説について述べる．計測および観察を通じて，食堂内の混雑や動線上の問題は，待ち行列そのものの形成だけでなく，座席やテーブルの配置，通路幅といった空間設計とも密接に関係している可能性が示唆された．

壁際の座席については，一方向からのみ利用している現在の配置を両側から利用可能な形に変更することで，座席数を増やせるのではないかという考えが議論された．一方で，単純に座席数を増やすことは，通路幅の縮小や動線の圧迫を招き，結果的に混雑をさらに悪化させる危険もある．こ

のことから，座席数の増加だけを目的としたレイアウト変更は不十分であり，利用者の移動経路と滞在場所のバランスを考慮した空間全体の設計が必要であると考えられるようになった．

また，現在の座席配置が混雑にどの程度影響しているのかは，観察だけでは必ずしも明確ではなかった．行列の位置や通路幅，テーブル間の距離が，衝突や回避行動，滞留の発生場所とどのように関係しているかを判断するためには，単に「混んでいるように見える」といった印象論ではなく，定量的な指標に基づいて検証する必要がある．そのため，本プロジェクトでは，レイアウトを変更したシミュレーション環境を構築し，衝突回数や人口密度といった行動指標を用いて比較することで，レイアウトの違いが混雑や動線に与える影響を明らかにすることを目指した．

以上より，レイアウトに関する仮説として，「通路幅や座席配置を調整することで，動線の交錯を減らし，食堂内の空間設計を改善できるのではないか」という方針を設定した．この仮説は，待ち行列の 2 列化という施策と組み合わせることで，入口付近と食堂内部の双方における混雑を総合的に緩和するための基盤となるものである．

（※文責: 西尾柊太）

第 6 章　数理モデルと数値計算手法

本章では，現実の食堂における複雑な混雑現象をシミュレーション上で再現するために用いた数理モデルと，それを実装するための数値計算手法について記述する．また，数理モデルを作成するにあたり，現実世界の事象を抽象化し，数学的な定義へと落とし込む必要がある．そこで，モデル化の前提となるエージェントの行動制約を整理している．

（※文責: 島津諒雅）

6.1　モデル化における条件や制約

本シミュレーションでは，現実世界の混雑時に見られる人同士の接触や押し合い，身体の圧縮，衣服同士の摩擦といった詳細な物理現象を，完全に再現しているわけではない．実際の群集では，密度が高くなるにつれて，人は他者と肩や腕が触れ合い，ときには押し戻されるような形で力を受けながら移動している．しかし，本プロジェクトにおける目的は，そのようなミクロな力学現象を忠実に再現することではなく，混雑状況における人流パターンや滞留領域の傾向を把握し，レイアウトや待ち行列配置の違いが人の流れに与える影響を評価することである．そこで，本シミュレーションでは，Social Force Model のパラメータ（希望速度や斥力の強さ，作用範囲など）を調整することで「衝突回避行動」を表現し，人と人が物理的に押し合う状況を，仮想的な力として近似的に扱うこととした．すなわち，エージェント同士が一定距離以内に近づいた際には，互いに離れようとする斥力が働くようにモデル化し，結果的に実際の混雑時に見られる「ぶつからないように避ける」行動を再現している．

また，本モデルにおけるエージェント間の相互作用は，基本的に衝突回避のための斥力のみに限定している．友人同士が横並びで歩行するグループ行動や，会話をしながら歩くことによる速度低下，「先にどうぞ」といった譲り合い行動，混雑を避けて遠回りを選択するような戦略的判断など，実際の食堂で観察される高度な社会的相互作用は考慮していない．これは，モデルの複雑さを抑え，レイアウトや待ち行列の構造と人流パターンの関係に焦点を当てるためであり，「人がどのような目的や関係性を持っているか」よりも，「空間構造がどこで流れを詰まらせるか」を重視した設計となっている．そのため，本シミュレーションから得られる結果は，あくまで「平均的な歩行者集団」が混雑空間を通過する際の傾向を捉えたものであり，特定の個人やグループ行動を対象とした詳細な分析には適していないという制約を持つ．

各エージェントの質量や反応時間などのパラメータについても，個体差は設けず，全エージェントで同一の値を用いている．現実には，年齢，性別，体格，歩行速度の好み，反応の速さなどに個人差が存在し，高齢者や荷物を持った利用者は歩行速度が遅くなったり，進路変更に時間がかかったりすることが想定される．しかし，本プロジェクトのシミュレーションは，個々人の違いを詳細に表現することよりも，「全体としてどのような人流パターンが形成されるか」「特定のレイアウトが混雑を助長するかどうか」を巨視的に把握することを主な目的としている．そのため，すべてのエージェントを「平均的な歩行者」として扱う単純化を行い，個体差によるばらつきは統計的ゆらぎの一部として切り捨てている．この単純化はモデルの解析性や実装の容易さを高める一方で，

「特定の条件下で一部の利用者が極端に不利になる」といった現象を捉えにくいという限界も併せ持つ．

空間構造の表現に関しても，現実の壁や障害物を完全な連続面として表現するのではなく，多数の微小な点（斥力点）の集合体としてモデル化している．具体的には，壁の輪郭に沿って一定間隔で点を配置し，それぞれの点がエージェントに対して斥力を及ぼすように設計することで，「壁から離れようとする力」や「障害物を避ける軌道」を生じさせている．このような表現方法を採用することで，直線的な壁だけでなく，曲がった通路や出入口付近のくびれ，カウンターやテーブル群など，複雑な形状を持つ障害物に対しても比較的柔軟に斥力計算を行うことが可能となっている．一方で，壁を完全な連続体として扱わないことにより，ごく近距離での挙動においては，実際の動きと多少異なる軌道を描く可能性があることにも留意する必要がある．

これらの前提条件や制約は，本プロジェクトのシミュレーションが「現実の状況をそのまま再現するための完全な仮想空間」ではなく，「混雑の発生傾向やレイアウトの違いによる影響を比較するための近似モデル」であることを意味している．したがって，本シミュレーション結果の解釈にあたっては，数値そのものを絶対的な予測値として扱うのではなく，「どの条件が相対的に混雑を増やすか，あるいは緩和するか」を判断するための参考指標として位置づけることが重要である．今後，モデルの高度化を図る場合には，個体差の導入やグループ行動のモデル化，心理的要因を反映した経路選択など，今回簡略化した要素を段階的に拡張していくことが検討課題となる．

（※文責: 菊地皓太）

6.2　数理モデル

作成した数理モデルは，シミュレーション空間内のエージェントに現実のような振る舞いを再現するためのものである．移動の物理的な法則と，意思決定の論理的なルールをを定義する必要がある．そのため，以下の理論を組み合わせてエージェントの挙動を記述する．

（※文責: 島津諒雅）

6.2.1　Social Force Model

Social Force Model は，歩行者の移動を物理的粒子の運動に見立て，ニュートンの運動方程式に基づき式に表すことができる．

Social Force Model を使った研究の例を紹介する．

Helbing ら (1995) は本モデルを用いて，駅構内や群集事故における人流の発生メカニズムやボトルネック現象を詳細に再現し，その有効性を示した．さらに Lakoba ら（2005）は，緊急時に特有のパニック行動を反映する項を導入し，危機状況下での人流挙動を精緻に表現可能とした．近年では Zhang ら（2019）により，実測データからモデルパラメータを推定し, 現実の群集行動を高精度に再現する研究も進展しており，Social Force Model の応用範囲は一層拡大している．本プロジェクトのシミュレーションにおいて，食堂内の歩行者の運動方程式は以下のように表した．

$$\boldsymbol{F}_{total} = \boldsymbol{F}_{drive} + \boldsymbol{F}_{rep,person} + \boldsymbol{F}_{rep,wall} \tag{6.1}$$

$\boldsymbol{F}_{drive}$ は目的地への希望推進力，$\boldsymbol{F}_{rep,person}$ は人からの反発力，$\boldsymbol{F}_{rep,wall}$ は壁などの障害物から

の反発力である．特に，希望推進力 $\boldsymbol{F}_{drive}$ は次のように表される．

$$\boldsymbol{F}_{drive} = m_i \frac{\boldsymbol{v}_i^0(t) - \boldsymbol{v}_i(t)}{\tau} \tag{6.2}$$

ここで，m_i は歩行者 i の質量，$\boldsymbol{v}_i^0(t)$ は歩行者の希望速度，$\boldsymbol{v}_i(t)$ は，歩行者の現在の速度，τ は速度調節の時間のスケールと定義する．次に，人からの反発力 $\boldsymbol{F}_{rep,person}$ は次のように表した．

$$\boldsymbol{F}_{rep,person} = \sum_{persons} A_1 \exp\left(-\frac{d_{ij}}{B_1}\right) \boldsymbol{n}_{ij} \tag{6.3}$$

A_1 は歩行者における反発力をスカラー倍する係数，d_{ij} は歩行者 i と歩行者 j との間の距離，B_1 は他の歩行者に対する減衰係数，$\boldsymbol{n}_{ij}$ は歩行者 j から歩行者 i に向かう単位ベクトルと定義する．次に，$\boldsymbol{F}_{rep,wall}$ は次のように定義した．

$$\boldsymbol{F}_{rep,wall} = \sum_{walls} A_2 \exp\left(-\frac{d_{iw}}{B_2}\right) \boldsymbol{n}_{iw} \tag{6.4}$$

A_2 は障害物における反発力をスカラー倍する係数，d_{iw} は歩行者 i と障害物 w との間の距離，B_2 は障害物に対する減衰係数，$\boldsymbol{n}_{iw}$ は障害物 w から歩行者 i に向かう単位ベクトルと定義する．

エージェントの運動はニュートンの運動方程式に従っており，これら 3 つの合力である $\boldsymbol{F}_{total}$ を用いて加速度 $\boldsymbol{a}(t)$ を次のように定義した．

$$\boldsymbol{a}(t) = \frac{\boldsymbol{F}_{total}}{m_i} \tag{6.5}$$

Helbing らが提案した Social Force Model には，人と人が直接的にぶつかったときに生じる physical force（物理的な力）がある．しかし，本プロジェクトで用いたモデルでは，人の大きさを考えないという単純化を行った．そのため physical force を無視し，心理的な衝突だけで考えることにした．

（※文責: 島津諒雅）

6.2.2　待ち行列モデル

本シミュレーションにおいて，エージェントが形成する待ち行列の待機座標 $\boldsymbol{P}_{target}(k)$ は，空間的な制約や運用条件の違いに基づき，以下の 3 つの数理モデルを用いて定義した．

１つ目は，混雑状況や運用ルールに応じて，行列の形成を一列とするか，二列とするかを選択可能なモデルである．

一列構成が選択された場合，エージェントは基準となる間隔を保ち，直線上に配置される．このときの k 番目のエージェントの待機座標は次のように表される．

$$\boldsymbol{P}_{target}(k) = \boldsymbol{P}_{head} + (d_{init} + (k-1)\, d_{gap})\, \boldsymbol{n} \tag{6.6}$$

ここで，$\boldsymbol{P}_{head}$ は行列の起点となる基準座標，d_{init} は基準点と行列の先頭エージェント間の初期オフセット距離，d_{gap} はエージェント間の標準距離間隔，$\boldsymbol{n}$ は行列の伸長方向を表す単位ベクトルと定義する．

二列構成が選択された場合，空間効率を高めるためにエージェント間の間隔を短くし，かつ順番 k が偶数か奇数かに応じて列の幅方向の位置をシフトさせる，

$$\boldsymbol{P}_{target}(k) = \begin{cases} \boldsymbol{P}_{head} + \left(d_{init} + k \cdot \frac{d_{gap}}{2}\right)\boldsymbol{n} + \boldsymbol{w} & k \text{ is odd} \\ \boldsymbol{P}_{head} + \left(d_{init} + k \cdot \frac{d_{gap}}{2}\right)\boldsymbol{n} & k \text{ is even} \end{cases} \tag{6.7}$$

ここで，$\boldsymbol{w}$ は列間のオフセットベクトルである．このモデルでは，標準間隔 d_{gap} の半分の間隔でエージェントを配置しており，奇数番目のエージェントに対して $\boldsymbol{w}$ を加算することで，ジグザグ状の配置を表現している．

２つ目は，通路幅等の物理的制約により，常に一列で整列するモデルである．

$$\boldsymbol{P}_{target}(k) = \boldsymbol{P}_{head} + (d_{init} + k \cdot d_{gap})\,\boldsymbol{n} \tag{6.8}$$

各変数の定義は前述と同様である．このモデルでは，$\boldsymbol{n}$ の設定により任意の方向への直線的な行列形成を表現することができる．

3 つ目は，待ち行列の待機スペースが限定的であり，行列長が一定の閾値を超過した場合に，動線を塞がないよう自動的に列をずらすモデルである．

$$\boldsymbol{P}_{target}(k) = \begin{cases} \boldsymbol{P}_{head} + k \cdot d_{gap}\boldsymbol{n} & k \leq K_{limit} \\ \boldsymbol{P}_{head} + k \cdot d_{gap}\boldsymbol{n} + \boldsymbol{v}_{shift} & k > K_{limit} \end{cases} \tag{6.9}$$

ここで，K_{limit} は第 1 列に許容される最大人数，$\boldsymbol{v}_{shigt}$ は列をずらすためのシフトベクトルと定義する．順番 k が閾値 K_{linit} を超えた場合，$\boldsymbol{v}_{shift}$ を加算することで第 2 列目を形成し，空間内での行列の形成を再現している．

（※文責: 島津諒雅）

6.3　数値計算手法

コンピュータ上でシミュレーションとして再現するためには，連続的な時間を微小な幅（タイムステップ）で区切り，離散的な計算処理として状態を更新していく必要がある．したがって，定義された数理モデルを近似的に解き，エージェントの位置や速度を時間発展させるための手法として採用している．

（※文責: 島津諒雅）

6.3.1　オイラー法

本シミュレーションでは，多数のエージェントをリアルタイムに近い速度で処理する必要があるため，計算コストと実装の用意さを考慮し，1 次精度の数値積分法であるオイラー法を採用した．

ニューロンの運動方程式により，時刻 t におけるエージェント i の加速度 $a_i(t)$ は，前節で求めた $\boldsymbol{F}_{total}$ を用いて次のように表される．

$$a_i(t) = \frac{\boldsymbol{F}_{total}(t)}{m_i} \tag{6.10}$$

この加速度を用いて，微笑時間Δ t 後の速度 $v_i(t + \Delta t)$ および位置 $p_i(t + \Delta t)$ を算出する．本シミュレーションでは，数値計算の安定性を高めるため，速度の更新を先に行い，その更新後の速度を用いて位置更新をする形式をとり，以下の差分方程式により状態を更新した．

$$v_i(t + \Delta t) = v_i(t) + a_i(t)\Delta t \tag{6.11}$$

$$p_i(t + \Delta t) = p_i(t) + v_i(t + \Delta t)\Delta t \tag{6.12}$$

ここで，Δt はシミュレーションの時間刻み幅（タイムステップ）であり，本計算ではアニメーションの描画更新間隔に基づき設定している．この反復計算を繰り返すことで，エージェントの連続的な移動軌跡を再現している．

（※文責: 島津諒雅）

第 7 章　シミュレータ開発

本章では，シミュレータのプロトタイプ開発やシステム開発プロセス，本開発についての詳細な内容を述べる．

（※文責: 菊地皓太）

7.1　プロトタイプ開発

シミュレータの本開発に先立ち，実現可能性を確認することを目的としてプロトタイプ開発を行った．プロトタイプは，食堂内の人流を「エージェントが空間内を移動する現象」として扱い，アニメーションとして可視化するための最小限の枠組みを構築する段階と位置付けた．

プロトタイプ開発の狙いは，完成形をいきなり作ることではなく，「シミュレータとして成立するために必要な要素（移動・回避・状態遷移・レイアウト反映など）は何か」を洗い出し，課題を顕在化させることである．

（※文責: 菊地皓太）

7.1.1　プロトタイプで実現できたこと

プロトタイプでは，主に以下の要素の実装や検証を試みた．

2 次元平面上でのエージェント可視化

食堂空間を 2D 座標として表し，エージェントを点として描画し，時間経過に合わせて位置が更新される形で可視化した．これにより，「人が動くシミュレーション」として成立するかを確認した．

目的地に向かう移動の実装

エージェントが入口から目的地へ向かって進む仕組みを導入し，主な行動（入場，移動，目的地への到達）を再現した．ここでは，目的地を座標として定め，そこへ向かう方向ベクトルに基づいて更新する方式を用いた．

障害物回避を用いた移動モデルの試作

壁やテーブルなどの障害物に対し，反発力のような仕組みを与えることで「ぶつからないで歩く」挙動を目指した．後の本開発で採用する Social Force Model の考え方に繋がる要素として，回避挙動の試作を行った．

食堂利用の状態遷移の一部の試行

人流の再現ではエージェントの移動だけでは不十分であり，提供・待ち行列に並ぶ・会計・着席・食器返却といった一連の行動が必要になる．そのため，プロトタイプ段階でも待ち行列や食事

提供，会計などの行動を再現すること試み，食堂利用の流れを表現するための課題を確認した．

以上よりプロトタイプは，可視化と移動を中心に最低限の機能の実装を行い，「食堂シミュレータを本当に作れるか」を確かめる段階としての役割を果たした．

（※文責: 菊地皓太）

7.1.2　プロトタイプで顕在化した技術的課題

一方で、プロトタイプ段階では本開発につながる複数の問題が顕在化した．特に重要だったのは次の 2 点である．

(1) 障害物回避（反発力）の不自然さと破綻

プロトタイプでは，矩形障害物に対する反発力を「障害物の中心点」から発生させる設計になっていた．この方式では，障害物全体の形状を正しく反映できないため以下のような問題が生じた．

・壁際や角付近で反発方向が不自然になる．

・角に引っかかって動けなくなることにより局所的停滞が起こる．

・障害物の縁をかすめる際に貫通や振動が発生する．

これは，実際には「障害物の中心」ではなく「障害物の境界形状」に応じて回避が生じるべきなのに，モデルがそれを表現できていないことが原因である．

(2) 最短経路移動による着席到達の破綻

プロトタイプでは，目的地（椅子など）へ向かう際に最短で直線に近い移動を行うため，食堂内のテーブル配置によっては以下のような問題が生じた．・椅子がテーブル付近にある場合に到達できない．

・目的地に向かって進むほど障害物に突っ込み続ける．

・回避と目的地到達が両立せず，停滞や無限ループに近い挙動になる．

これは食堂のように障害物が多い環境では，単純な最短志向だけでは到達可能な経路が確保できないことを示している．

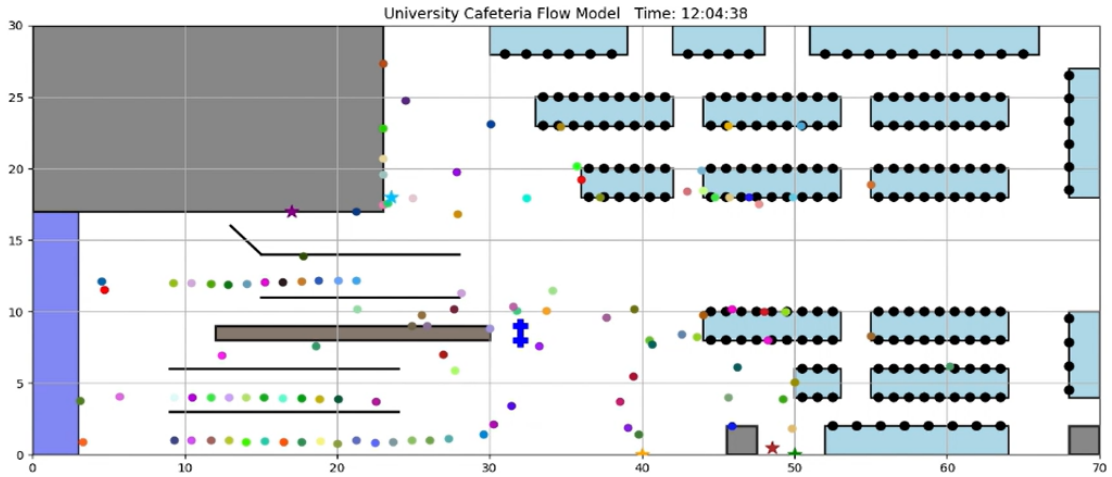

図 7.1　プロトタイプのシミュレーション画面

（※文責: 菊地皓太）

7.1.3　プロトタイプで顕在化した開発進行上の課題

技術的課題に加え，開発の進め方にも課題があった．プロトタイプは「0 から 1」を作る段階で試行錯誤が多く，結果として次の問題が残った．

全体像の把握に時間を要した．

システム開発経験が少ない状態で着手したため，必要機能の整理や依存関係の把握に手間取り，設計や実装の流れが安定しなかった．

ファイル分割が過剰になり，変更影響が追跡しづらかった．

分割の必要性が薄い段階で複数ファイルを作成し，同時並行で修正した結果，どの変更がどの挙動に影響したかが分かりづらくなった．

バグを残したまま機能追加を優先し，問題が複雑化した．

不具合がある状態で次の機能を継ぎ足したため，原因の切り分けが困難になり，結果として修正コストが増大した．

実装手順の整理が不足していた．

「何を順番に作るべきか」や「どこまでできたら次へ進むか」といった手順が曖昧で，到達目標と作業内容がぶれやすかった．

（※文責: 菊地皓太）

7.2　システム開発プロセス

シミュレータの開発は，プロトタイプ作成，本開発の二段階で進め，本開発の事前準備として，システム開発プロセスを要件定義，基本設計，詳細設計の順に整理を行った．また，実装は小さな機能単位で段階的に追加し，動作確認と修正を繰り返すことで，仕様の安定化と品質確保を図った．

（※文責: 菊地皓太）

7.2.1　プロトタイプ開発の振り返りと本開発の方針

プロトタイプ開発では，シミュレータの「0 から 1」を作る過程で試行錯誤が多く，開発の進め方に課題が残った．

・システム開発経験が少ない状態でプロトタイプ構築に着手したため全体像の把握に時間を要したこと．

・分割の必要が薄い複数ファイルを作成して同時並行で修正し，変更影響が追いづらくなったこと．

・バグが残った状態で次の機能追加を優先し，問題が複雑化したこと．

・実装手順の整理が不足していたこと．

などが具体例として挙げられる．

これらの反省を踏まえ，本開発では次の方針で開発を進めた．

目標と進捗の可視化：毎回「やりたいこと／今回やること」を明確にし，作業の優先順位と到達点を揃える．

可読性重視の実装：後から見返しても理解できるよう，変数名・関数分割・コメントを含めて分か

りやすいコーディングを徹底する.
段階的な品質確保：バグを保留せず，原因を特定して修正してから次へ進む.

（※文責: 菊地皓太）

7.2.2 本開発の導入目標

本開発では，いきなり多数のエージェントや複雑な確率イベントを導入せず，まずエージェント 1 人が入場し，食事提供・会計・着席・返却を経て退場する一連の流れを実現することを直近目標とした.

1 人がゴールするという目標方針により，状態遷移の妥当性，レイアウトと経路設計の不整合，障害物回避の破綻などを早期に発見できる．その後，1 人で安定動作することを確認した上で，複数人化・待ち行列・確率イベントを段階的に追加し，混雑現象の再現性を高めていく方針とした.

（※文責: 菊地皓太）

7.2.3 要件定義

要件定義では，プロトタイプで顕在化した問題点を整理し，「本開発で必ず解決すべき要件」として明文化した.
障害物回避（反発力）の実装改善：プロトタイプでは矩形障害物に対する反発力が中心点から発生しており，壁際・角付近で挙動が不自然になる課題があった.
着席動作の実現：プロトタイプでは最短経路で移動する結果，テーブルに阻まれて椅子へ到達できないケースが発生した.

また，エージェントが「入場から退場までに取る行動」を仕様として定義し，食堂利用の実態に近づけるために，待機・立ち止まり・確率イベントを含めて段階化した.

・入場
・食事提供
・会計
・食事準備（箸の取得，ウォーターサーバーへの立ち寄り）
・着席
・食器返却
・退場

（※文責: 菊地皓太）

7.2.4 基本設計

基本設計では，シミュレータが備えるべき機能を整理し，評価実験（混雑緩和策の比較）に必要な入出力と操作性を含めて設計した.

1. 機能要件

エージェントの可視化と移動モデル：Social Force Model を用い，障害物や他エージェントを回避しつつ目的地まで移動する．また，待ち行列の形成や座席滞在といった混雑に直結する状態を表

現する.
レイアウトの反映：食堂マップ上に障害物や各種オブジェクトを反映し，テーブルや椅子のレイアウト変更にも対応する.
データ入力：日付や時間別の入退場人数など，Excel で集計したデータを読み込めるようにする.
GUI / アニメーション制御：停止・再生・時間スキップ機能等を備え，混雑時の挙動を観察しやすくする.
2. 非機能要件
仮想時間：実時間を圧縮して再生することで検証を効率化する.
再現性：同一パラメータで同一結果が得られる再現性を確保し，施策比較の公平性を担保する.
ユーザビリティ：障害物・テーブル・椅子等を直感的に理解できる表示とし，挙動解釈を容易にする.
セキュリティ：ファイルやシミュレーション結果に個人情報を含めない.

（※文責: 菊地皓太）

7.2.5 詳細設計

詳細設計では，実装を責務ごとに分割し，保守性（修正しやすさ）と拡張性（機能追加のしやすさ）を確保した．全体は概ね「パラメータ設定」「レイアウト描画」「エージェント行動」「メイン制御」の 4 層で構成する.
1. パラメータ設定：混雑状況の再現性と施策比較の公平性を確保するため，シミュレーションの挙動に影響する値をパラメータとして一括管理し，実験条件の切り替えや調整を容易にした．パラメータは役割に応じて以下のカテゴリに整理し，シミュレーション実行時に参照される.
2. レイアウト定義と描画：壁・棚・テーブル等は，線分や矩形として定義し，描画関数で可視化する．また，レジ等のオブジェクトは座標で位置を定義し，Matplotlib の描画で表現する方針とした.
3. 障害物回避：壁は「矩形」と「線分」で表現し，障害物周辺に回避計算用の点（斥力点）を配置する．これにより，障害物の中心点だけでなく周辺形状に応じた回避が可能となり，壁際・角付近での貫通や引っかかりを低減できる.
4. 待ち行列：提供待ち・会計待ち・返却待ち・ウォーターサーバー待ち等は，サービス地点に対してキュー構造を導入する方針とした．エージェントがサービス地点へ到達した時点で利用中なら末尾に追加し，先頭が処理完了して離脱したら残りが前詰めで更新される処理として実装した.
5. 中継点と座席到達のための経路設計：食堂内は障害物が多く，目的地へ直線移動すると衝突や滞留が発生しやすい．そこで「中継点（waypoints）」を導入し，セット・カレー・ラーメン等の各ルートに対して「提供，会計，着席，返却，退場」を含む移動系列を用意した．中継点はルートごとにリストで管理し，座標だけでなく各中継点での待ち時間も持たせることで，立ち止まり行動を実装できるようにした．また，座席到達については，プロトタイプで「最短経路の結果，テーブルに阻まれる」という問題があったため，椅子位置を通路側へオフセットし，さらに座席用の中継点を設定して到達性を改善した.

（※文責: 菊地皓太）

7.2.6　開発体制

本開発は複数名で実施し，機能ごとに担当を置きつつ，相互レビューと統合を繰り返す形で進めた．運用としては，毎回の作業前に「やりたいこと／今回やること」を定め，進捗を共有することで，作業の重複や手戻りを抑制した．

また，開発中は「分かりやすいコーディング」と「バグを保留しない」方針を徹底し，安定動作を維持したまま機能追加できる体制を整えた．加えて，実装支援として生成 AI（Gemini）を活用し，機能追加ごとにその時点のコードを保存して履歴を残す運用を行った．これにより，不具合が生じた場合でも直前の安定版へ戻って原因切り分けができ，段階的な改良を進めやすくした．

（※文責: 菊地皓太）

7.3　シミュレータ概要

本シミュレータは，大学食堂内における人流を，複数の来客をエージェントとして扱うエージェントベースモデルによって再現し，混雑や衝突の発生状況を観察・比較することを目的に作成した．このシミュレータを利用してできることを本質的に言えば，「現実の食堂のレイアウトと来客データを入力として，人が目的地へ向かって歩く過程を 2 次元平面上で再現し，その結果として生じる混雑を可視化する」ということである．

具体的には，食堂内を 2 次元平面としてモデル化し，壁・通路・オブジェクト等を配置する．そして，来客を Customer エージェントとして生成し、各エージェントには「入場する」「料理を受け取る」「着席する」「退出する」といった一連の行動を与える．ただし，エージェントは単に決められた動線上を移動するのではなく，目的地へ向かう駆動力と、壁・障害物・他者との距離を保つ斥力（Social Force に基づく力）を組み合わせて移動する．これにより，人が通路で自然に避け合ったり，狭い箇所で詰まったりするといった現象が，計算結果として自然に生じる．

また，本シミュレータは現実の食堂の入退場者数を記録した Count.csv に記された時間帯ごとの来客数データを用いて，特定の時刻に来客が集中する状況を再現している．来客は分単位の人数に従って生成され，同じ 1 分間の中ではランダムなタイミングで入場するため，実際の食堂に近い形で入店のばらつきも表現できる．このようにして，時間帯によって混雑がどのように変化するか，また混雑が発生するとどこで衝突・滞留が起きやすいかの観察が可能である．

（※文責: 菊地皓太）

7.4　シミュレータ実装

本章では，本研究で開発したシミュレータの実装内容や実装方法について述べる．

（※文責: 菊地皓太）

7.4.1 主要パラメータ

ここでは，シミュレータの実装において必要となる各種パラメータのうち，特に挙動や結果に大きく影響する主要なものを抜粋して示す．これらのパラメータは，エージェントの移動速度や目的地到達判定，障害物・他者回避の強さ，および滞在時間など，シミュレーション全体の動作を決定づける要素である．また，各値を調整することで，混雑の発生傾向や人流の自然さを変化させることが可能である．

シミュレーション全体（時間・更新）
VIRTUAL __ TIME __ SCALE：仮想時間の進み方（= 速度調整）
SIMULATION __ START __ TIME：CSV 基準の開始時刻（例 “1130”）
ANIMATION __ INTERVAL：描画更新間隔 (ms)（= フレームレートに効く）
frames __ per __ virtual __ minute = (1000 / ANIMATION __ INTERVAL) * VIRTUAL __ TIME __ SCALE：「仮想 1 分あたり何フレームか」を決める式で、入場スケジュール計算や時刻表示の基準になります。

エージェント挙動
AGENT __ DESIRED __ SPEED：目標速度
TARGET __ REACH __ THRESHOLD：ターゲット到達判定距離
WAYPOINT __ RANDOMNESS：中継点にランダムオフセットを入れて動きを自然にする半径
EXIT __ SPEED __ MULTIPLIER：最後のゴール（退出）に向かうときの速度倍率
MAX __ SPEED __ MULTIPLIER：目標速度に対する最大速度の上限倍率

着席・ウォーターサーバー利用
MIN __ CHAIR __ WAIT __ SECONDS, MAX __ CHAIR __ WAIT __ SECONDS：着席滞在時間の範囲
WATER __ SERVER __ PROBABILITY / POST __ SEATING __ WATER __ SERVER __ PROBABILITY：立ち寄り確率
CHAIR __ SEARCH __ START __ INDEX：経路の何番目以降で椅子探索を始めるか

Social Force 系（力学パラメータ）
RELAXATION __ TIME：目標速度へ追従する速さ（小さいほど素早く速度を合わせる）
REPULSION __ FORCE __ STRENGTH, REPULSION __ FORCE __ DECAY, REPULSION __ EFFECTIVE __ RANGE：壁・障害物の斥力
SOCIAL __ FORCE __ STRENGTH, SOCIAL __ FORCE __ DECAY, SOCIAL __ FORCE __ EFFECTIVE __ RANGE：他エージェントの斥力

（※文責: 菊地皓太）

7.4.2 描画処理

ここでは，本シミュレータにおける可視化処理を担う描画関数について説明する．描画関数は，食堂内の壁・机・椅子などの静的オブジェクト，および Customer エージェントの位置や移動の様子を Matplotlib 上に表示する役割を持つ．

静的オブジェクト（机・椅子・壁・障害物）
テーブル：Rectangle，facecolor=’lightblue’（薄青）
椅子：Circle，デフォルト color=’black’（黒）
壁線：Line2D で黒線（color=’black’）
障害物矩形（SPACES __ RECT など）：灰色半透明（facecolor=’gray’, alpha=0.5）
斥力点（壁や矩形境界上に置く点）：Circle で黒（facecolor=”black”）
壁や障害物の周囲に点をばら撒き，その点群を使って壁斥力を計算する構造である．

ルート（中継点の可視化）
Route 1/2/3 の中継点：scatter の marker=’x’
色：orange，green，purple（3 ルートで色分け）

設備（星マーカー）
例としてコードに明示されているもの：
ゴミ箱：color=’brown’，marker=’*’
テラス出入口：color=’orangered’，marker=’*’

エージェント
Customer エージェント：Circle パッチ（色はエージェント毎にランダム RGB）
その人が向かっている椅子の表示：半径 0.35 の Circle を用意し，エージェントの色で塗って表示（edgecolor=’black’）

衝突表示
衝突地点マーカー：赤い x（color=’red’, marker=’x’）＋衝突回数テキスト（赤）
衝突判定は「着席していないアクティブ同士」で距離が AGENT __ RADIUS 未満のときにカウントし，一定時間表示・クールダウンも含まれる．

表示切り替え（チェックボックス UI）
右側チェックボックスで，テーブル/椅子/オブジェクト/斥力点/各ルート/椅子用中継点/向かう椅子/衝突情報の ON/OFF 切替が可能である．

（※文責: 菊地皓太）

7.4.3　エージェント実装

ここでは，本シミュレータにおいて来客を表現するエージェントの実装について説明する．本シミュレータでは，各来客を Customer エージェントとしてモデル化し，それぞれが独立して移動や行動を行うように設計されている．エージェントは位置や速度といった物理量に加え，目的地や行動状態（移動中，着席中，退出中など）を内部に保持し，周囲の環境や他エージェントとの相互作用に基づいて次の動作を決定する．このようなエージェント実装により，個々の行動の積み重ねとして全体の人流や混雑が発生する様子を再現している．

初期化（init）
id，color，pos，velocity を持つ構成となっている．
ルートは all __ waypoints（3 ルートの集合）からランダムに 1 つ選び，最後にゴールを追加して targets とする．
entry __ frame（登場フレーム）を持ち，is __ active=False から開始する．
着席関連フラグ（is __ seated，target __ chair __ pos，chair __ target __ added,，eturn __ route __ added 等）を持って状態管理を行う．

力の計算（calculate __ forces）
待機中・ゴール済みは力 0 となる．
駆動力 F __ driving：目標方向へ向かう速度（desired __ velocity）へ現在速度を近づける力．
壁斥力 F __ rep：壁（正確には「斥力点群」）との距離が有効範囲内のとき，指数減衰で押し返す．
この後ろに他エージェント斥力（F __ social）が続き，入場直後は一定時間無視するという設計となる．

中継点進行・待機・座席探索
ターゲット到達後，ターゲットに wait があれば wait __ timer をセットし，待機中は速度 0 にする（特に椅子は座標固定も行う）。
target __ index == chair __ index で，椅子を確保できるなら AVAILABLE __ CHAIRS から 1 つ受け取り，座席エリアに応じた中継点列を挿入して座席へ向かう．
椅子が満席なら，以降を切ってゴールへ直行する．
椅子に座ったあとウォーターサーバーへ行く確率分岐も含む．

（※文責: 菊地皓太）

7.4.4　メイン関数について

ここでは，本シミュレータ全体の処理を統括するメイン関数について説明する．メイン関数では，Count.csv から来客数データを読み込み，使用するレイアウト（Original，A，B）やシミュレーション条件を設定した上で，エージェントの生成およびアニメーション描画を開始する．また，フレーム更新ごとにエージェントの出現タイミング管理，移動・状態更新，退出処理などを実行し，シミュレーションを時間経過に沿って進行させる役割を担う．このようにメイン関数は，入力データの処理から可視化までを接続する中心的な部分であり，シミュレータ全体の動作フローを

管理する重要な構成要素となっている.

読み込み・シミュレーション実行部分
Count.csv を読み込む．読み込むファイルが存在しなければエラーとなる.
CSV のヘッダから日付一覧を作成し，テスト用データも反映する.
各レイアウト（Original/A/B），待ち行列タイプ（一列/二列），日付を RadioButtons で選択し，開始ボタンで run __ simulation() を呼ぶ.

メイン処理部分
大きく 3 段階構成となっている.
(1) レイアウトを展開
レイアウト辞書から，入口・出口・壁・机・椅子などをグローバル変数へ読み込む.
(2) 入場スケジュール（entry __ frames）を作成
CSV を時刻行毎に読み，各分の人数 num __ entrants の分だけその 1 分に相当するフレーム範囲へランダム割当を行う.
これで，何フレーム目に誰が入場するかのリストが完成し，NUM __ AGENTS = len(entry __ frames) になる.
(3) エージェント生成とアニメーション更新
Customer を entry __ frame 付きで全員作成し色もランダム RGB で反映させる.
init() で描画を初期化し，update() で毎フレーム更新する.
指定フレームになったら is __ active=True にして出現させる.
力を計算して移動・状態を更新する.
ゴール到達したらリストから削除する.
タイトルに仮想時刻を表示する.
加えて，時刻ジャンプ機能として「指定時刻のフレームまで描画なしで早送りして状態を作り直す」処理も含む.

（※文責: 菊地皓太）

7.4.5 各種レイアウトデータ実装

ここでは，本シミュレータにおける各種レイアウトデータの実装について説明する．本シミュレータでは，食堂内の机・椅子・壁・通路・設備位置などをレイアウトデータとして定義し，その内容を切り替えることで異なる空間配置を再現できるようにしている．具体的には，現状の食堂配置を再現した Original を基準として，比較検討のために机や椅子配置を変更した Layout A および Layout B を用意し，同一条件でシミュレーションを行うことでレイアウト変更が人流や混雑に与える影響を評価可能としている.

layout.py の基本構造
get __ layout(layout __ name) が，共通データ＋選択レイアウト固有データをまとめた辞書を返す.
共通として入るもの：

COMMON __ POSITIONS（入口/出口/返却口などの座標）
COMMON __ WALLS __ LINE，COMMON __ SPACES __ RECT
WAYPOINTS __ 1/2/3（セット/カレー/ラーメンのルート）

Original/A/B の差分
レイアウト名で分岐し，以下を差し替える．
TABLES __*：机の矩形配置
CHAIR __ POSITIONS __*：椅子座標群
CHAIR __ WAYPOINTS __*：椅子エリアへ行くための中継点辞書（A は Q1..、B は R1.. のようなキー）
CHAIR __ AREA __ PATTERNS __*：椅子の位置範囲ごとに「どの中継点列を使うか」のパターン集合
RETURN __ ROUTE __ PATTERNS __*：座席から復路の中継点列パターン（座席位置で分岐）
WATER __ SERVER __ RETURN __ PATTERNS __*：ウォーターサーバーから座席に戻るパターン
最後に AVAILABLE __ CHAIRS を「そのレイアウトの椅子一覧からシャッフルしたもの」として作成し，辞書に入れて返す．

（※文責: 菊地皓太）

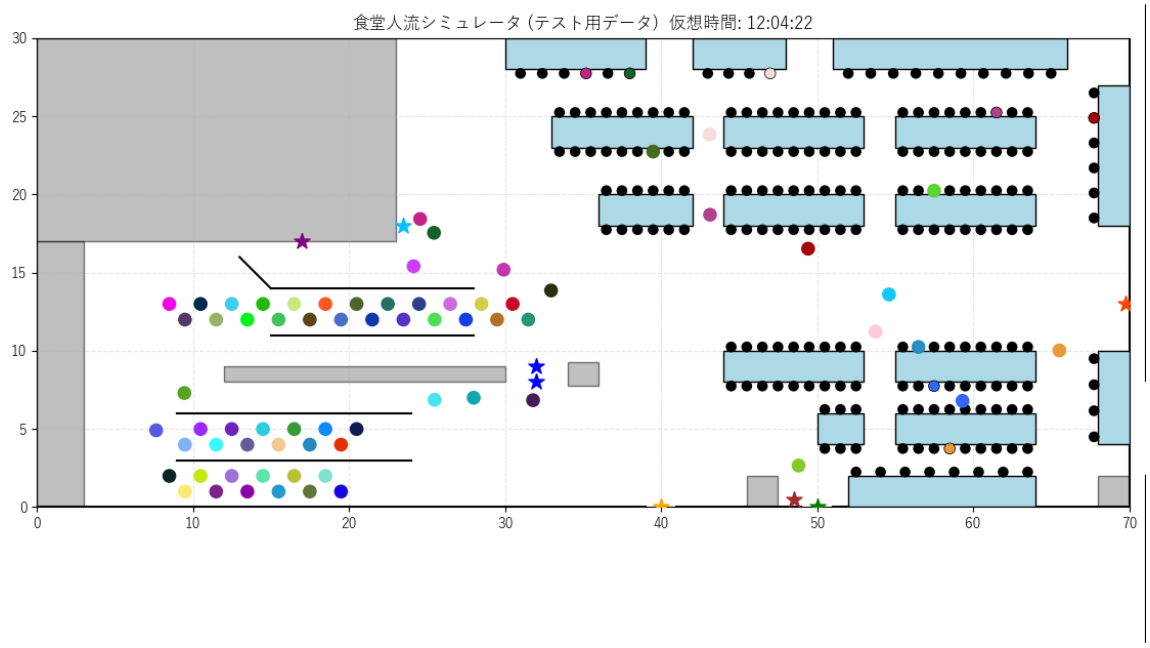

図 7.2　シミュレーション画面

第 8 章　シミュレーションに基づく検証

本プロジェクトは，大学食堂における混雑緩和を最終目的とし，その効果予測のためにエージェントベース型のシミュレーションを実施した．本章では，混雑状況を定量的に評価するために設定した指標の振り返り，シミュレーション条件，および混雑緩和策の提案について述べる．

（※文責: 櫻井孔士）

8.1　検証

検証に先立ち， 本研究では混雑を定量的に評価するための指標を定義した． 混雑の指標として，「衝突回数」「人口密度」の 2 要素を採用した．

1：衝突回数

エージェント同士が一定距離以下となり，衝突判定が発生した総数．混雑による移動阻害，歩行快適性の低下，動線干渉の度合いを示す指標として位置づける．

2：人口密度

対象領域（特に出入口周辺）に一定時間内に滞留する人数を記録した値．空間負荷・滞留の偏りを評価する指標として扱う．

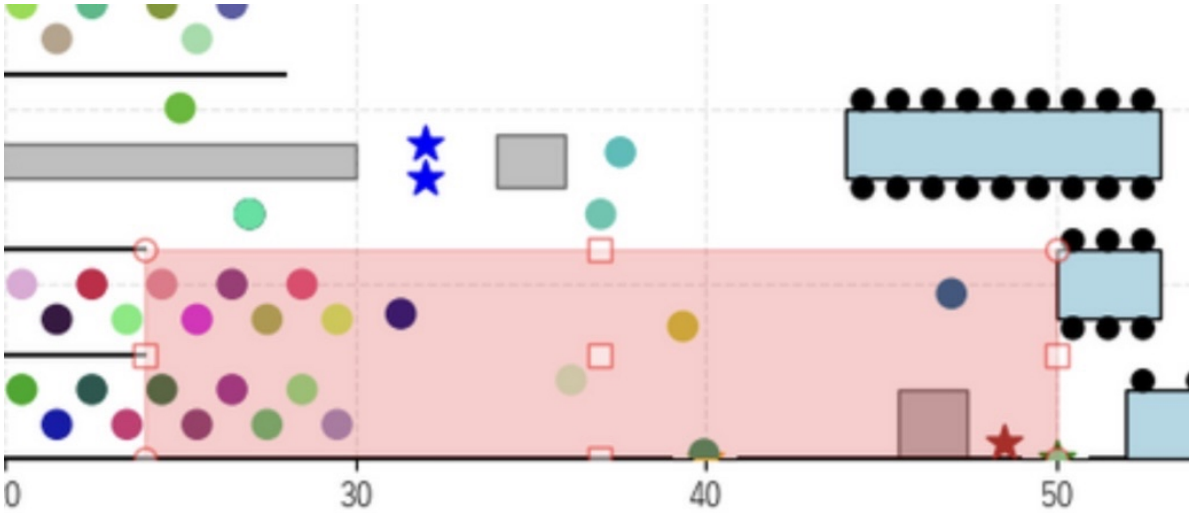

図 8.1　人口密度検証の対象領域

衝突回数は動線設計の妥当性を，人口密度はレイアウト設計の有効性を評価する指標となるという役割分担を意図した．

また，検証は開発した食堂シミュレータを用いて実施した． 対象時間帯は 12：00〜12：30 に限定し，入力データには検証用データを用いた． 検証用データは，計測データのうち金曜日のデータのみを抽出し，その平均をとって作成した． 上記条件のもとで，衝突回数および出入り口付近の人口密度を測定した． 検証条件は，レイアウト 3 種類 （Layout __ original，Layout __ A，Layout __ B） と，待ち行列形状 2 種類 （1 列，2 列） の組合せからなる計 6 通りとした．

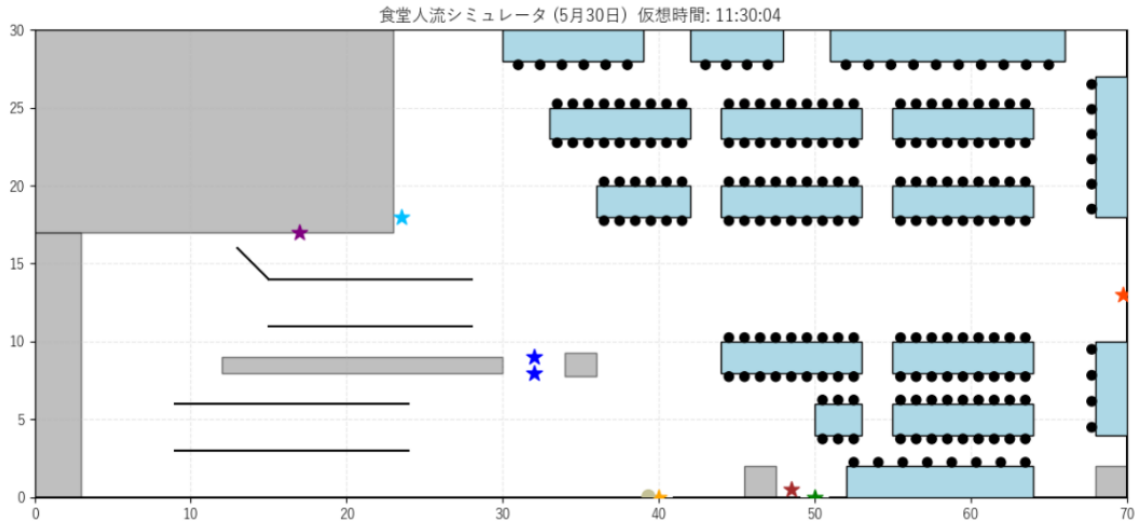

図 8.2　Layout __ original

Original レイアウトは，現在の食堂の実際の配置（現状レイアウト）を再現した基準モデルとして位置づけられる．椅子の配置や通路幅などを現実の食堂に合わせて再現することで，シミュレーション結果が現実に対応しうる状態を作っている．この Original レイアウトを基準として，机や椅子の配置を変更した比較用レイアウトを作成し，それらを同一条件（来客数データやエージェント挙動）でシミュレーションを行うことで，レイアウト変更が混雑・衝突・滞留などに与える影響を視覚的に比較検討することを可能とする．

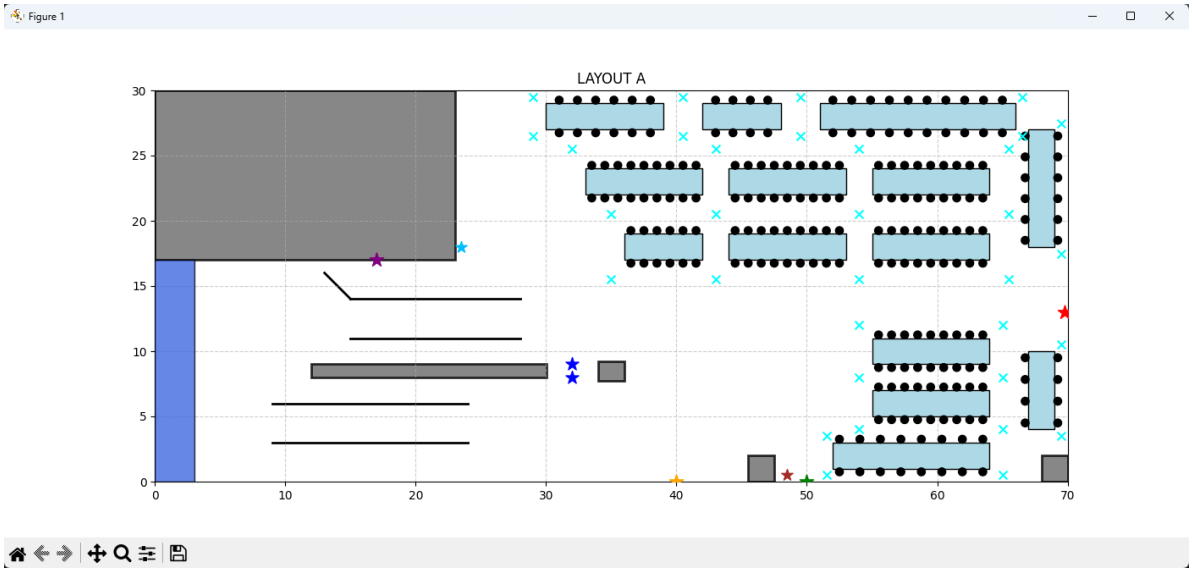

図 8.3　Layout __ A

変更点：Layout __ original 内で壁際のテーブルを壁から離し，両側から座れるように座席を増やした．それに伴って，増えた座席数分出口近くのテーブル 2 つを撤去した．これにより，で入口付近にスペースが生まれ動線を確保する狙いがある．

変更点：Layout __ A 内における出口付近の 3 つのテーブルを再配置をした．Layout __ A で出口付近に生まれたスペースにテーブルを一つ移動させる．これによってさらに生まれたスペースにテーブルを詰め，壁際にスペースができるようにする．これにより，セットの料理提供待ち行列が

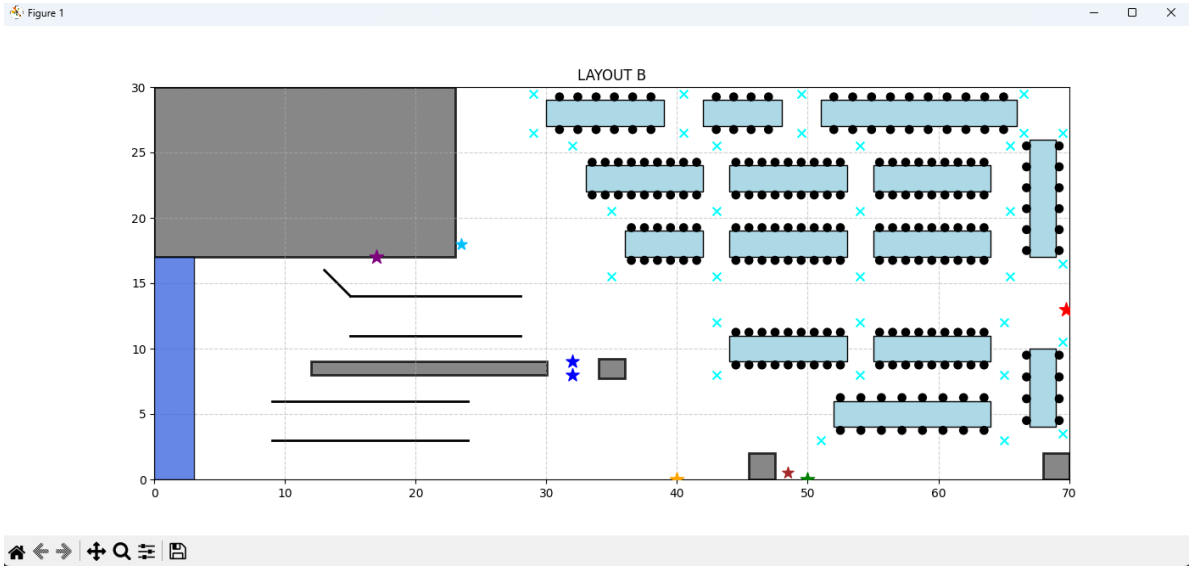

図 8.4　Layout __ B

長大化した際，行列の延長線上にスペースが存在しているため，出口付近のテーブルでの混雑を解消する狙いがある．

（※文責: 櫻井孔士）

8.2　検証結果

各条件の結果を比較したところ，待ち行列を 1 列から 2 列に変更すると，多くのレイアウトで衝突回数が減少した． さらに，2 列条件では衝突発生位置が出入り口付近に集中する傾向が確認された． 衝突回数の測定結果を表 8.1 に示す．

表8.1　衝突回数の比較

衝突回数	一列	二列
Layout_original	194	165
Layout_A	237	172
Layout_B	151	179

一方で，人口密度については，レイアウト間で顕著な差は確認されず，レイアウト変更による人口密度の大きな変化は見られなかった. 各レイアウトと待ち行列形状の組み合わせによる人口密度の測定結果を以下に示す．

（※文責: 櫻井孔士）

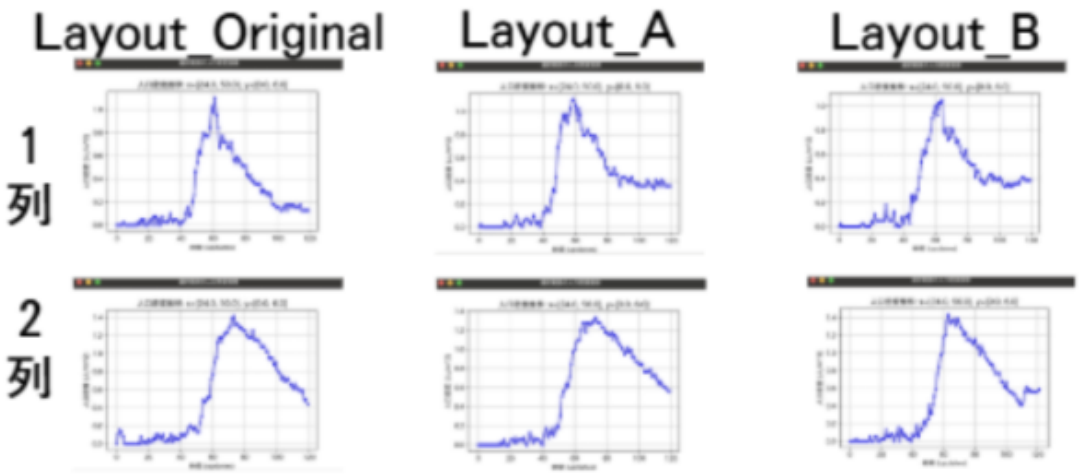

図 8.5　人口密度比較

8.3　考察

本検証においてまず注目すべき点は，待ち行列を 1 列から 2 列へ変更した場合に，出入り口付近の人口密度が増加した理由である．待ち行列を 2 列化すると，行列全体の奥行きが短縮され，食堂内部の通路やテーブル周辺への圧迫は軽減される傾向が見られた．しかしその一方で，行列自体は出入り口付近に集約され，利用者が一定時間その場に滞留する結果となった．そのため，待機列が持つ「列の膨張」を食堂内部に押し込める形から，「入口の手前で保持する形」へと負荷の位置が移動し，結果として出入り口付近に局所的な密度上昇が生じたと考えられる．つまり，2 列化が内部空間の混雑抑制に寄与する一方で，その代償として出入り口付近での混雑を増幅させる構造が生まれたと推察される．

次に，Layout __ B において 2 列化によって衝突回数が増加したという結果について考察する．1 列条件ではテーブル配置が動線を分散させる効果を持ち，エージェントが複数方向へ拡散することで接触が分散した可能性がある．これに対して 2 列条件では，行列が出入り口付近に強く集中し，利用者がその狭い領域内を通過しようとする際に回避行動が困難となる状況が生じたと考えられる．その結果，衝突が特定エリアに偏在し，全体の衝突回数として増加が確認されたと解釈できる．このことは，待ち行列の本数を増やすこと自体が必ずしも改善に直結するわけではなく，「どこに負荷が移動し，どの領域で混雑が増幅するか」という視点を伴った設計が不可欠であることを示唆している．

最後に，2 列条件における各レイアウト間の衝突回数には大きな差が見られなかった点について考察する．これは，2 列化によって衝突発生領域が出入り口付近に限定され，レイアウトの違いが影響を及ぼしやすい食堂内部の動線，すなわちテーブル間の通行幅や回遊経路といった要素の寄与が相対的に小さくなったことが原因と推測される．そのため，待ち行列が 2 列で運用される状況下では，レイアウトによる改善効果は限定的となり，根本的課題は内部空間よりも出入り口付近に存在していると考えられる．以上より，本検証は，単純なレイアウト変更による改善には限界があり，出入り口周辺での立ち位置誘導や行列の形成位置といった運用的要素が，混雑緩和における決定的な支配因子となる可能性を示している．

（※文責: 櫻井孔士）

8.4 提案

以上の考察結果を踏まえると，特に Layout __ B におけるレイアウト改善は依然として有効性を持つと考えられる．Layout __ B ではテーブル配置が動線の分散を促し，1 列条件ではテーブル周辺の衝突抑制につながる特徴が確認された．この構造的特性は，待ち行列を 2 列化した際にも一定程度維持され，食堂内部の歩行空間を確保する上で寄与し得るものである．ただし，2 列条件下では出入り口周辺の局所混雑が増加するという課題が明らかとなっているため，単にレイアウトを変更するだけでなく，行列の起点位置の調整，入退場動線の明確化，立ち位置の視覚誘導といった運用面での対策を併用する必要がある．

したがって，今後の改善案としては，レイアウトと運用対策の両面からアプローチすることが望ましい．具体的には，Layout __ B を基盤としつつ，出入り口付近に緩衝領域を設け，行列が内部へ侵入しないよう導線を設計することで，内部と入口の両方に配慮した混雑制御が期待される．このような改善策は，単に混雑を縮小させるのではなく，「混雑がどこに発生し，どのように制御すべきか」という視点に基づき，混雑の偏在を抑制しながら歩行快適性の向上を図る取り組みとして妥当であると考えられる．

（※文責: 櫻井孔士）

第 9 章　中間発表

本節では中間発表のための準備，中間発表当日，頂いたフィードバックの評価について振り返る．なお，中間発表時点では，食堂における入退場者数の測定や，利用者へのアンケート，問題点設定，Social force モデルや 1 列待ち行列の実装や時間ごとの食堂内の人流モデルの表示といった段階まであった．

（※文責: 鈴木創太）

9.1　中間発表の準備

本プロジェクトでは，中間発表に向けてスライド資料とポスター資料の 2 種類を作成し，発表媒体として活用した．スライドは発表の主軸となる説明資料として構成し，問題提起から調査手法，モデル選定の理由，そして試作段階のシミュレーション結果へと論理的に展開できるよう意識して制作した．一方でポスターは，聴衆が立ち止まって閲覧した際に理解を補助できるよう，スライドで触れきれない背景設定や用語の補足説明，人流問題の図示化などを中心に配置したものである．これにより，発表中はスライドを通じて聞き手の理解を誘導しつつ，発表後の質疑や個別説明の際にはポスターを参照しながら議論を進められるような二層的な説明構造を整えた．

発表準備を進めるうえで最も重視した点は，説明の内容を欲張りすぎないことであった．中間発表の段階では研究全体の最終結論が出ているわけではなく，調査や検証が未完了の部分も存在している．そのため，本発表では「どこまで検証が進んでいるのか」「何を明らかにできており，何が今後の課題として残るのか」という線引きを明確に記述し，一貫したストーリーとして提示することを意図した．特に，食堂の混雑問題の中でも待ち行列に焦点を絞るという判断は，聞き手がテーマの全体像を把握しやすくするための工夫であり，研究対象の輪郭を明示する役割を果たしたと考えられる．以上の理由から，中間発表に向けた資料構成は，今後の研究の進め方や調査方針の方向づけとしても重要な意義を持つものであった．

（※文責: 鈴木創太）

9.2　中間発表の内容

中間発表の構成は，大きく 5 つの段階に整理した．まず最初に数理モデリングという手法そのものについて説明を行い，本プロジェクトが「単なる混雑の感覚的議論」ではなく，定量的分析に基づく問題解決を目指していることを明示した．そのうえで，本学食堂における混雑問題を提示し，現地観測と計測，およびアンケート調査によって「問題が実際に存在すること」を前提条件として成立させた．次に，本プロジェクトで採用した Social Force Model と待ち行列モデルについて説明し，シミュレーションがどのような前提とルールに基づいて動作しているのか，行動原理の枠組みを示した．

また，作成したシミュレーションの試作版を実演し，基礎的な挙動が再現されていることを示した．具体的には，6 月 23 日（月）から 6 月 27 日（金）までに計測した 1 分ごとの入場者数グラ

フ，アンケート結果の可視化資料，Social Force Model と待ち行列モデルの実行の様子などを用いて説明した．このように，「現実の観測」「数理的な解釈」「仮想空間での再現」という流れを意識した発表構成とすることで，シミュレーションが単なる映像生成ではなく，現実に基づき再現可能性を持った検証基盤であることを示すことができたと考えられる．

（※文責: 鈴木創太）

9.3　当日の動き

中間発表は 2025 年 7 月 4 日（金）に公立はこだて未来大学にて実施され，本プロジェクトは前半時間帯に割り当てられた．発表は全 3 回行い，メンバーそれぞれが担当を分担しながら交代で登壇した．それぞれプロジェクト内で発表順を決めており，1 回目の発表では西尾が主発表者として説明を行い，菊地がタイムキープと質疑時の返答を担当した．2 回目は島津が発表を務め，櫻井が質問内容の記録と補助的回答を担った．3 回目では鈴木が口頭説明と質疑応答を担当し，坂井が時間管理と質問記録を行った．これらの発表体制は，1 人が全てを背負うのではなく，チーム全体として発表の完成度を高めるという姿勢を反映したものであり，発表後の振り返りにおいても有意義であったとの評価が共有された．発表は当初の予定通りスライドとポスターを用いた発表を行った．

（※文責: 鈴木創太）

9.4　発表の評価

発表終了後，聴衆からのフィードバックを得るために Google フォームを用いた評価アンケートを実施した．回答者は学生および教員合わせて 27 名であり，そのうち学生が 77.8 %，教員が 22.2 %を占めた．評価項目は「発表技術」と「発表内容」の 2 種類を 10 段階評価で尋ねたものであり，平均値は発表技術が m=7.7（SD=1.3），発表内容が m=7.8（SD=1.5）と，全体として中程度から高評価に位置する結果となった．

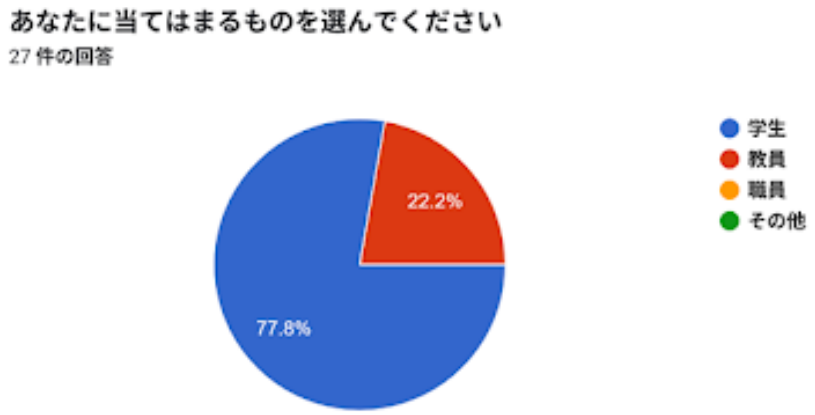

図 9.1　中間発表の評価者の種別

（※文責: 鈴木創太）

図 9.2　中間発表の発表技術についての評価

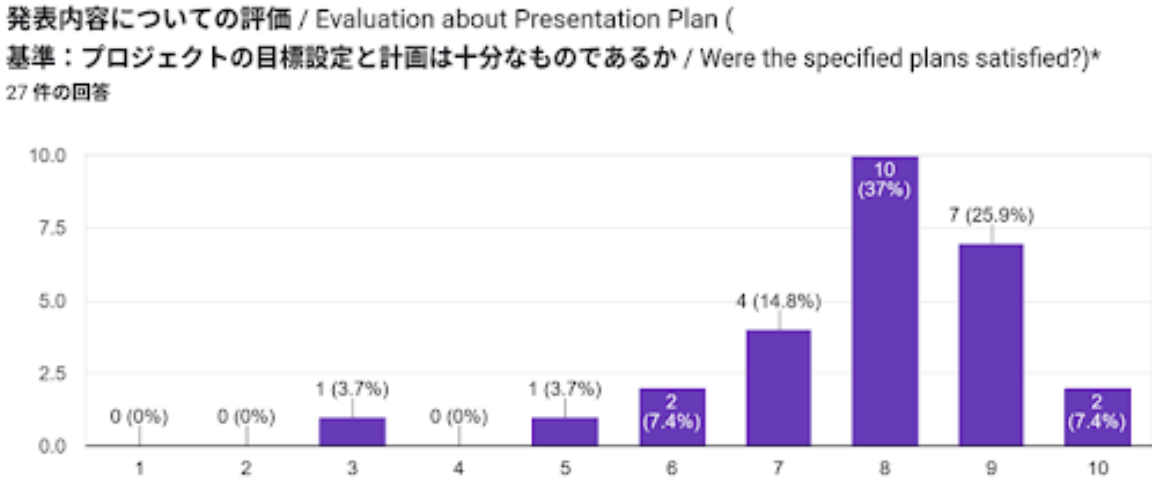

図 9.3　中間発表の発表内容についての評価

9.5　発表技術コメント

発表技術に関していくつかのフィードバックが寄せられた．寄せられたフィードバックをここに列挙する．

ポジティブな意見

・スライドが見やすかった．

・わかりやすい説明だった．

アドバイス

・早口で聞き取りにくい．

ポジティブとアドバイスが混在してる意見

・説明は分かりやすかったが，もう少し遠くに聞こえるよう意識して欲しい．
・呼びかけといった工夫は良かったが不明瞭な部分もあった．

（※文責: 鈴木創太）

9.6　発表内容コメント

9.5 同様，発表内容についても寄せられたコメントを列挙していく．
ポジティブな意見
・難しいことを図やデータを用いて解説してくれていてよかった．
・アプローチがしっかりできていると思いました．
・シュミレーションを組み込んでいて分かりやすかった．
アドバイス
・数式や図の各項目がどのような意味なのかわからない．
・人流の改善で，本当に混雑が解消するかの疑問はある．
ポジティブとアドバイスが混在してる意見
・流れが分かりやすいが解決策をもう少し説明して欲しい．
・モデルの中身についてはもう少し詳しい説明が必要．

（※文責: 鈴木創太）

9.7　評価の考察と改善策

フィードバック全体を総括すると，スライド構成や視覚提示方法については一定の評価が得られた一方，音声的伝達力や説明の速度調整といった非言語的要素が課題として浮き彫りになった．特に，「資料は理解しやすいが，音声情報による補助が追いついていない」という指摘は複数見られ，発表者と資料の情報提示量のバランスに課題が残ることが明らかとなった．

また，発表内容に関する指摘として，「モデルの提示は理解できたが，最終的にどのような介入策や政策的示唆につながるのか」という疑問が挙げられた．これは，中間発表という研究進行段階ゆえに起こり得る指摘であるものの，今後の研究方向性を説明する際の構造が弱かった可能性を示唆している．したがって，今後は「現状把握」「モデル化」「検証」「解決策の適用」という構造に沿ったロードマップを示し，研究の目的意識を明確にする必要があると考えられる．

以上より，次回発表や最終報告に向けては，発声訓練や原稿調整による伝達技術の改善と，研究の方向性を提示するための説明設計の再構築が必要である．発表練習の回数を増やし，擬似的なリハーサル環境を整えることで改善できる余地は大きく，今後の取り組みによって発表品質がさらに向上することが期待される．

（※文責: 鈴木創太）

第 10 章　成果発表

本節では成果発表のための準備，成果発表当日，頂いたフィードバックの評価について振り返る．

（※文責: 鈴木創太）

10.1　成果発表の準備

成果発表にあたり，本プロジェクトでは中間発表と同様にスライドとポスター資料（2 枚）を作成した．ただし今回は，単なる活動報告ではなく「成果として提示できる論理構造」を形成することを意識し，資料制作の段階から内容の取捨選択と再整理を行った．スライドは発表全体の導線となる説明資料として位置づけ，背景・目的・モデル化の方針・レイアウト別検証の根拠・結果と考察・今後の展望という流れが自然に理解できるよう設計した．特に成果発表で新たに加えた要素として，仮説生成の根拠とシミュレーションの結果が研究上どのような意味を持つのかを明示し，「活動」から「検証可能な知見」へ移行したことを伝える点を意識した．

一方ポスターでは，聴衆が立ち止まって目を引くようなデザインで，モデルとなる数式やレイアウト比較図，行列 2 列化の意図や人口密度の解釈など，スライドでは説明しきれない背景情報を補った．すなわち，スライドが「説明を聞いて理解する資料」であるのに対し，ポスターは「読み返して納得できる資料」として機能させることを目標とした．この役割分担により，発表中と発表後の両面に対応できる説明体制を整えることができたと考えられる．

（※文責: 鈴木創太）

10.2　成果発表の内容

成果発表においては,「なぜこの方法を選び，どのように検証し，何が明らかになったか」を聴衆に伝えることを目的として構成を再設計した．まず食堂の混雑が発生する背景とデータ収集によって確認された問題点を提示し，現象を感覚的認識ではなく数値的根拠に基づいて捉える必要性を示した．

また，本プロジェクトで作成したレイアウト（Layout __ Original，Layout __ A，Layout __ B）について紹介し，それぞれの違いが歩行の動線，待ち行列の形成，衝突発生にどのような影響を与えるかを提示した．さらに，人口密度と衝突回数という評価指標を用いた検証結果を示し，行列 2 列化によって「衝突箇所の局所集中」が生じる可能性があることや，レイアウト変更のみでは抜本的な混雑緩和に至らない可能性を考察として示した．最後に，今後の展望として「行列位置指定」「誘導ルール導入」「レジ前処理短縮策の検証」といった運用改善への応用可能性を述べ，研究の継続性と発展性を示した．

（※文責: 鈴木創太）

10.3　当日の動き

成果発表は 2025 年 12 月 5 日（金）に公立はこだて未来大学にて実施され，本プロジェクトは前半時間帯に割り当てられた．発表は全 3 回行い，メンバーそれぞれが担当を分担しながら交代で登壇した．それぞれプロジェクト内で発表順を決めており，1 回目の発表は西尾，菊地が担当した．西尾はタイムキーパー兼質問のメモを担当し，菊地は口頭での発表と質問への返答を行った．2 回目の発表は鈴木，坂井が担当した．鈴木はタイムキーパー兼質問のメモを担当し，坂井は発表と質問への返答を行った．3 回目の発表は櫻井，島津が担当した．櫻井はタイムキーパー兼と質問のメモを担当し，島津は発表と質問への返答を行った．これらの発表体制は，1 人が全てを背負うのではなく，チーム全体として発表の完成度を高めるという姿勢を反映したものであり，発表後の振り返りにおいても有意義であったとの評価が共有された．発表は当初の予定通りスライドとポスターを用いた発表を行った．

（※文責: 鈴木創太）

10.4　発表の評価

聴衆には QR コードから Google フォームへアクセスしてもらい評価を収集した．回答者は学生・教員合わせて 27 名であり，学生が 66.7 %，教員が 27.8 %であった．評価項目は「発表技術」と「発表内容」の 2 軸で行われ，平均値は，発表技術が m=6.9（SD=2.5），発表内容が m=8.2（SD=1.2）となった．

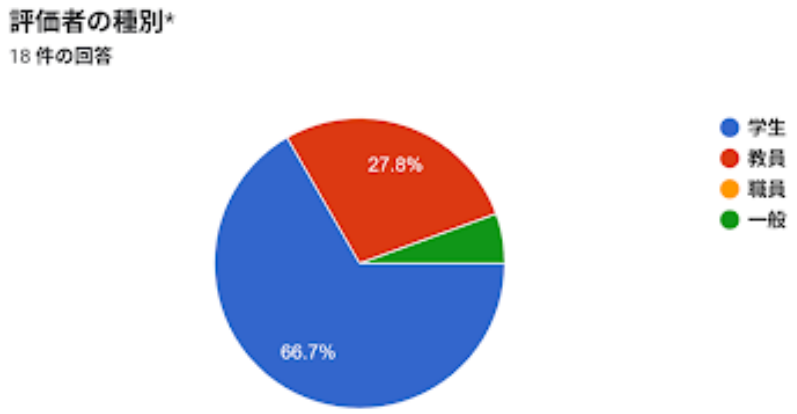

図 10.1　成果発表の評価者の種別

（※文責: 鈴木創太）

10.5　発表技術コメント

発表技術に関していくつかのフィードバックが寄せられた．寄せられたフィードバックをここに列挙する．

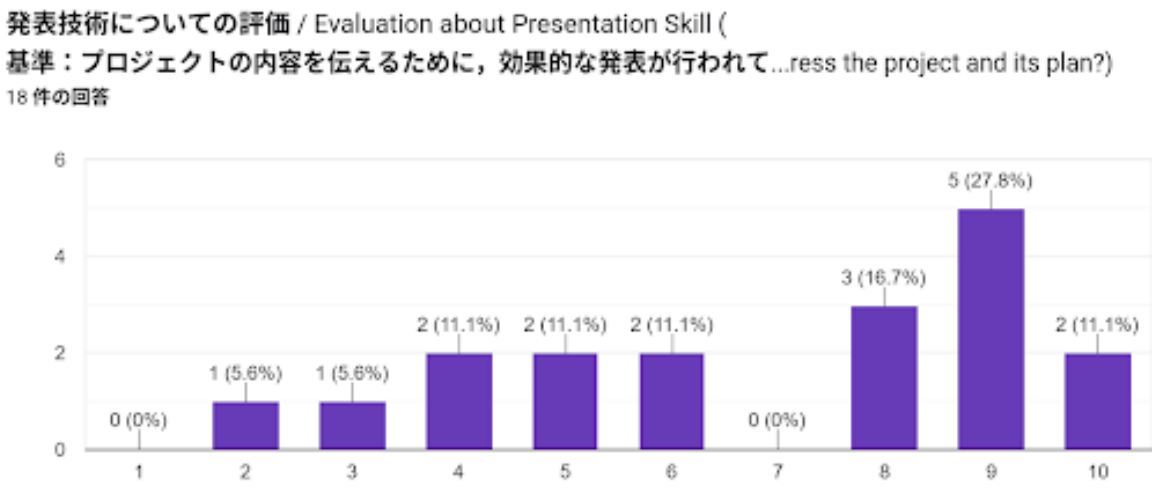

図 10.2　成果発表の発表技術についての評価

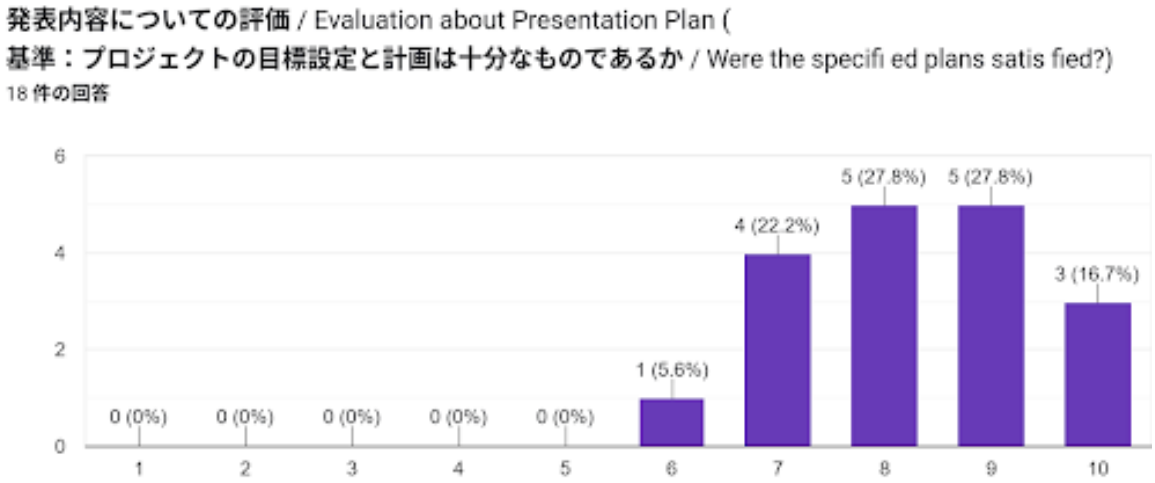

図 10.3　成果発表の発表内容についての評価

ポジティブな意見

・スライドがまとまっていてわかりやすかった.

・モデリングが精巧で面白かった.

アドバイス

・少し声が小さくて聞こえにくい時があったから声を大きくしてほしい.

・聴衆側に向けて話す方がいい.

ポジティブとアドバイスが混在してる意見

・スライドがとても見やすいです．もう少し大きな声で発表できるとよかったです.

・発表の声を聞き取ることはできなかったが，スライドに図表が効果的に使われていたので、内容を理解することはできた.

・スライドは見やすいものの、声の小ささが気になった.

（※文責: 鈴木創太）

10.6　発表内容コメント

10.5 と同様，発表内容についても寄せられたコメントを列挙していく．

ポジティブな意見

・テーマ決めが上手だと思いました．

・食堂を利用する側なのでとても面白かったです．

・スライドと説明が分かりやすい．

アドバイス

・プロジェクトにおいて数理モデルが介入するところが知りたい．

ポジティブとアドバイスが混在してる意見

・身近な課題解決について数理的な方法でアプローチしている点がよいと思いました．

・二列化とレイアウト変更というソリューションをどう思いついたのか知りたかったです．

（※文責: 鈴木創太）

10.7　評価の考察と改善策

提出されたフィードバックを総合すると，本プロジェクトの強みは「資料設計と説明内容の整理」，弱みは「口頭発表時の伝達技術」にあることが再確認された．特に，発表資料に対しては「構造化されていて理解しやすい」「図やデータを用いた説明が有効であった」といった意見が多く，準備段階での工夫が成果につながっていると評価できる．一方で，「二列化やレイアウト変更という解決策に至る思考過程を知りたい」「数理モデルとの関連付けが十分に言語化されていない」という指摘は，研究の意義説明とアプローチ提示の不足を示している．

以上を踏まえると，今後の改善策として，単に声量やスピードを修正するだけではなく，「論理展開を支える言語化力」「研究選択の理由を説明する技術」を強化する必要がある．また，結論だけでなく「なぜその施策を試す価値があるのか」という因果関係を示すことで，数理モデリングを導入する意義がより伝わりやすくなると考えられる．最終的には，モデル構築の結果を現場への改善提案へ接続できるアウトプット力の向上が，今後の課題であり展望であると言える．

（※文責: 鈴木創太）

第 11 章　まとめ

本章では，本プロジェクトにおける活動の総括と今後の展望について述べる．

（※文責: 坂井康大）

11.1　活動の総括

本プロジェクトでは，未来大学の食堂における混雑状況を対象とし，混雑の可視化および混雑緩和策の提案と検証を目的として活動を行った．そのために，実地調査による現状把握から数理モデルに基づくシミュレーションの構築，ならびに条件を変えた検証までを一連の流れとして実施した．

はじめに，食堂の実態を把握するためのデータ収集を行った．具体的には，食堂内のレイアウト測量，時間帯ごとの入退場者数の計測，利用者および従業員を対象としたアンケート調査を実施した．入場者数の計測結果からは，食堂営業開始直後および 2 限終了直後に利用者が集中する傾向が確認された．また，利用者アンケートの結果，行列や待ち時間を問題点として挙げた回答が約 65 %を占めており，ピーク時には待ち行列が食堂の端まで伸び，出入口付近の動線が悪化していることが明らかとなった．

次に，収集したデータを基に混雑緩和策に関する仮説を設定した．待ち行列の観点からは，行列を二列化することで入口および出口付近の動線を確保できると考えた．また，レイアウトの観点からは，通路幅や座席配置を調整することで空間利用の改善が可能であると仮定し，2 種類のレイアウト案を提案した．

これらの仮説を検証するため，エージェントモデルに基づく食堂シミュレータを構築した．シミュレータでは，来店者をエージェントとして扱い，「移動」「着席」「退席」といった行動を時間発展的に再現できるよう設計した．数理モデルとしては，歩行者の相互作用を表現する Social Force Model および待ち行列モデルを採用した．

検証では，12 時から 12 時 30 分までの時間帯を対象とし，衝突回数および出入口付近の人口密度を混雑の指標として測定した．レイアウト 3 通りと待ち行列 2 通りを組み合わせた計 6 条件でシミュレーションを行い，結果の比較を行った．その結果，二列の待ち行列では行列長が短縮され，衝突回数が減少するとともに，衝突の発生位置が出入口付近に集中する傾向が確認された．一方で，人口密度についてはレイアウトによる大きな差は見られなかった．

（※文責: 坂井康大）

11.2　今後の展望

本プロジェクトでは，主に現状把握と基礎的な影響検証に留まったが，今後の展望としては以下の 3 つの方向性が示唆される．

1 つ目：行列制御と動線管理の最適化

出入口付近の衝突リスクを低減するため，行列の並ぶ位置の床面表示やパーティションの設置と

いった「物理的誘導」，および混雑情報を提示する「情報的誘導」の併用が有効と考えられる．特に，行列の制御が入口・出口の交錯を抑制できることがシミュレーション上で確認できたため，現場での試験導入と事前検証の両輪による評価が望まれる．

2 つ目：提供フロー・会計処理の効率化

行列の長大化そのものを抑制するには，単なる空間改善では不十分であり，提供プロセスの高速化が求められる．食券制の導入，セルフ会計機の設置，生協アプリ準備の促進施策（ポスター・QR 案内など），人員配置の最適化などをモデルに組み込むことで，運用改善の費用対効果を事前に評価できる枠組みが構築可能である．とりわけ，「配置人数を 1 名増やした際の効果」を可視化するシミュレーションは，人的コスト判断の根拠として現場導入に繋がり得る．

3 つ目：モデル精度の向上とシステム化

現状モデルは基礎的な人流挙動を再現する段階に留まっているため，次の拡張が有効と考えられる．

・利用者属性（個人差，速度差，目的地選択）の導入
・時間帯変動モデルの導入による非定常シミュレーション化
・環境変化（天候，講義時間変更，イベント実施）への適応検証

これらの拡張により，単なる研究モデルではなく，「運用改善の意思決定支援ツール」への発展が期待される．

今後は，本プロジェクトで示された改善案を基盤とし，実験的な現場導入とシミュレーション結果の照合を反復することで，食堂運営にとって実装可能性の高い施策を検討していきたい．本成果が，大学施設運用における混雑緩和の一助となり，同様の課題を抱える他施設への応用展開につながることを期待する．

（※文責: 坂井康大）

謝辞

本プロジェクトの遂行にあたり，多大なるご支援と貴重なご意見を賜りました専務理事の宮田隆生様ならびにアンケート・インタビューにご協力いただいた皆様に，心より感謝申し上げます．皆様のご協力に深く御礼申し上げます．

（※文責: 菊地皓太）

参考文献

[1] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, Vol. 51, No. 5, pp. 4282–4286, 1995.

[2] T. I. Lakoba, D. J. Kaup, and N. M. Finkelstein. Modifications of the helbing-molnar-farkas-vicsek social force model for pedestrian evolution. *Simulation*, Vol. 81, No. 5, pp. 339–352, 2005.

[3] J. Zhang, W. Klingsch, A. Schadschneider, and A. Seyfried. Transferring pedestrian movement characteristics from laboratory experiments to real-world scenarios. *Transportation Research Part C: Emerging Technologies*, Vol. 19, No. 3, pp. 494–506, 2011.

[4] 西成活裕. 渋滞学. 新潮社, 2006.

[5] 橋本修, 毛塚敦. Python による数値計算法の基礎. 森北出版, 2021.

[6] 三井情報株式会社. シフトスケジューリングの最適化とは？課題やメリット、解決方法を紹介. MSIISM. https://www.msiism.jp/article/solution-scheduling.html.

[7] 山下拓三. 室内被害シミュレーションの利活用研究 -防災教育のための vr 可視化一. 国立研究開発法人 防災科学技術研究所 地震減災実験研究部門, 2018. https://www.bosai.go.jp/hyogo/research/symposium/pdf/20180329_5.pdf.

[8] 石濱新乃輔, 久瀬瑛利花, 河路友也, 藤田美和子, 中山浩. 人流と外気温による換気量と空調機出力制御による省エネルギー効果検証. 空気調和・衛生工学会中部支部学術研究発表会論文集, No. 25, pp. 51–54, 3 2024. https://www.jstage.jst.go.jp/article/shasec/25/0/25_51/_pdf/-char/ja.

[9] 長山雅晴. 数理モデリングとは何か. 数学通信（日本数学会）, Vol. 28, No. 1, 2023. https://www.mathsoc.jp/publications/tushin/backnumber/index23-4.html.

[10] 藤野俊樹, 増田知昭. スーパーマーケットで客はどう動く？一顧客動線分析とエージェントシミュレーションからわかること一. 東京工業大学. https://mas.kke.co.jp/wp-content/uploads/2020/01/thesis.pdf.

付録 A　付録

本付録では，本文中で扱いきれなかった補足事項や詳細な解説をまとめる．

（※文責: 菊地皓太）

A.1　専門用語

本付録では，本報告書で使用した専門用語および特有の語句について，理解の補助を目的として解説を行う．本文中で用いた略語や数理モデル，解析手法に関する用語についても併せて整理し，必要に応じて簡潔な定義を示す．

（※文責: 菊地皓太）

A.1.1　数理モデリング / 研究プロセス関連用語

数理モデリング

数理モデリングとは，現実世界で起きている現象や課題を，数学的な表現（変数，数式，規則，確率など）によって記述し，その理解や予測，改善策の検討に活用する一連の取り組みである．ここで重要なのは，単に式を作る行為そのものではなく，「現実の問題を分析し，必要な要素を抽出し，目的に沿った形に整理する」という思考過程が中心にある点である．例えば食堂の混雑問題であれば，混雑が起きる要因を洗い出し，それらの関係を表現できるように変数化する必要がある．

一般に数理モデリングは，(1) 問題設定，(2) 現象の観察とデータ収集，(3) モデル化，(4) パラメータ設定，(5) シミュレーション，(6) 検証や改善，という段階を踏む．このとき，「モデルを一度作ったら終わり」ではなく，得られた結果を現実と照らし合わせ，必要に応じてモデルの仮定やパラメータを見直す反復過程を含む点が特徴である．

また，現実は極めて複雑であるため，モデルは必ず何らかの近似を含む．したがって，数理モデリングでは「どれだけ現実を忠実に再現できたか」よりも，「目的に対して必要十分な精度で本質的な構造を捉えているか」が重要となる．そのため，結果を解釈する際には，前提条件や制約条件を明確にし，モデルが扱える範囲を意識して用いる必要がある．

数理モデル

数理モデルとは，対象となる現象を数学的に表現したものであり，基本的には「変数」と「その変数同士の関係」を明示することで構成される．変数は観測・計測・推定できる量として定義され，関係は数式，規則，確率過程，またはアルゴリズムとして与えられる．例えば食堂の混雑を扱う場合，来客数，入店時刻分布，到着率，提供の処理時間，移動速度，通路幅，席数などが変数になり得る．そして，それらがどのように待ち時間や行列長，滞留の発生に影響するかを関係として記述する．

数理モデルには複数の種類があり，現象を大きな単位で捉えるマクロモデル（待ち行列理論の M/M/1 など）と，個々の主体の動きを扱うミクロモデル（Social Force Model やエージェント

ベースモデル）に大別できる．マクロモデルは簡潔で計算が容易な一方，個々の行動差や動線の交錯などの詳細を表しにくい．ミクロモデルは詳細な挙動を表現しやすい一方で，必要なパラメータが増え，計算量も大きくなる．

さらに，数理モデルは作成者の仮定に依存する．したがって，モデルの結果を現実の事実のように扱うのではなく，「仮定の下での予測・比較結果」として解釈する姿勢が必要である．そのためには，モデル内で何を無視し，何を重要視したのかを明確にし，適用可能な状況を示すことが重要となる．

定式化

定式化とは，現実の問題や現象を数学的に扱える形へ翻訳し，数式や規則として表現可能な構造に落とし込む作業である．現実の課題は通常，複数要因が絡む曖昧な形で存在するため，そのままでは計算や比較検討ができない．そこで定式化を行い，「何を変数とみなし」「どの関係をモデルに組み込み」「どの量を評価対象とするか」を明確化することで，分析可能な形に整える．

例えば混雑を扱う場合，「混雑している」という状態を，平均待ち時間，最大待ち時間，行列長，滞留人数，通路密度などの具体的指標として定義する必要がある．さらに，それらが時間とともにどう変化するかを表すために，到着率やサービス率，移動速度などを導入し，式やルールとして関係づける．また，複数の改善案を比較するなら，評価すべき目的関数を明示し，その比較基準を設定する必要がある．

定式化の難しさは，現象を説明するために必要な要因を取り込みつつ，過度に複雑にならないように調整する点にある．要因を省きすぎると現実を説明できず，取り込みすぎるとパラメータが増え，検証や計算が困難になる．したがって定式化では，「目的にとって本質的な要因を残す」という設計思想が重要となり，その根拠を文章で説明できる形にする必要がある．

抽象化

抽象化とは，現実の複雑な現象から，目的にとって重要な構造だけを取り出し，それ以外の詳細を意図的に省略する操作である．数理モデリングは現実そのものを複製する作業ではないため，必ず抽象化が含まれる．例えば食堂の混雑には，利用者の性格差，歩行の癖，会話による滞留，気分，天候など無数の要因が影響し得るが，すべてを取り入れることは不可能であり，また必ずしも必要ではない．そこで，目的に直接関係する要因（到着率の集中，窓口数，処理時間，動線の交錯など）を優先して取り込む．

抽象化が適切であると，モデルは簡潔になり，計算も容易になり，結果の解釈も明確になる．さらに，現象の本質的メカニズムを理解しやすくなる利点がある．一方で，抽象化が過度であると，現実とのずれが大きくなり，評価を誤る危険が生じる．例えば，利用者の滞留行動や迷い行動を無視すると，実際には発生する通路詰まりがモデルでは再現できない可能性がある．

したがって抽象化では，「何を捨て，何を残したか」を明示することが重要である．その上で，抽象化によって生じる限界を理解し，結論の範囲を適切に設定する必要がある．

因果関係

因果関係とは，ある要因（原因）が別の事象（結果）を引き起こすという関係である．現象の理解や改善策の検討において，因果関係の把握は中心的役割を果たす．例えば「到着率が増えると行列長が増える」「窓口数を増やすと待ち時間が減少する」「動線が交錯すると滞留が増え処理効率が落ちる」といった関係は，因果関係として理解できる．

ただし現実では，多数の要因が同時に変動するため，「相関がある＝因果がある」とは限らない点に注意が必要である．例えば雨の日に混雑が増えたとしても，雨が混雑を直接生むとは限らず，別の要因（屋内利用の集中など）が介在している可能性がある．したがって因果関係を議論するには，背景要因や前提条件を考慮し，モデル内での仮定を明確にする必要がある．

数理モデルでは，因果関係は式やルールに組み込まれる．つまり，モデルの出力は「仮定した因果関係を前提にした結果」である．そのため，モデル結果を根拠として示す場合は，「どの因果を仮定し」「その因果が現実に妥当と考えた理由」を説明することで，説得力が高まる．

意思決定支援

意思決定支援とは，複数の選択肢の中から適切な方針を決める際に，判断材料を整理し，根拠を与えることで意思決定を助けることである．食堂運営の例では，窓口増設，レイアウト変更，導線分離，案内表示の改善，人員配置変更などが選択肢になり得る．しかし，現場ではすべてを同時に試すことは難しく，施策にはコストや制約が伴う．そこで数理モデルやシミュレーションによって，「どの施策がどの程度効果を持つか」を事前に比較できるようにすることが意思決定支援となる．

意思決定支援の目的は「唯一の正解を自動的に出すこと」ではなく，施策の効果・副作用・トレードオフを可視化し，関係者が合理的に議論できる土台を提供することである．例えば待ち時間は減ったが通路混雑が増えた，回転率は上がったがスタッフ負担が増えた，などの側面を示すことで，現実的な判断が可能になる．

また，意思決定支援では，不確実性の取り扱いも重要である．データ不足やモデル仮定による誤差が存在する以上，結果を断定するのではなく，条件依存性や限界を示したうえで判断材料として提示する姿勢が求められる．

評価指標

評価指標とは，モデルの結果や施策の効果を比較・判断するための「物差し」となる量である．適切な評価指標を定めることで，改善案同士を客観的に比較でき，議論が感覚論に偏ることを防ぐことができる．食堂混雑の例では，平均待ち時間，最大待ち時間，行列長，滞留人数，窓口処理人数，座席回転率，通路の密度などが評価指標となる．

評価指標を選ぶ際に重要なのは，「研究目的と整合しているか」である．例えば目的が「利用者のストレス軽減」なら平均待ち時間が重要となるが，「安全性向上」なら通路密度や動線交錯の頻度が重要となり得る．指標が目的とずれていると，改善案を誤って評価する危険がある．また，1つの指標だけで判断すると偏りが生じるため，複数指標を併用し，施策の長所短所を総合的に判断することが望ましい．

さらに，評価指標は算出方法を明確にする必要がある．例えば「待ち時間」を並び始めから受け取り完了までとするのか，レジ到達までとするのかで値は変化する．

定量評価

定量評価とは，数値で表される指標を用いて，施策やモデルを比較・評価する方法である．数値化されることで差が明確になり，結論の説得力を高めやすいという利点がある．例えば平均待ち時間が 10 分から 5 分に減少した，最大行列長が 30 人から 20 人に減少した，処理人数が一定時間で増加した，などの形で示される．

定量評価では，単に値を示すだけでなく，算出条件をそろえたうえで比較することが不可欠であ

る．到着率やシミュレーション時間，利用者行動ルールが異なれば結果は変化するため，同一条件下で施策のみを変更して比較する必要がある．また，乱数を含むモデルでは試行ごとに結果が変動するため，複数回試行し平均や分散を示すことで評価の安定性が高まる．

一方で，定量評価は「測れない価値」を扱いにくい．例えば利用者の分かりやすさや満足感，スタッフの負担感といった側面は数値化が難しい．したがって，定量評価は有力な方法であるが，必要に応じて定性評価と併用し，多面的な判断を行うことが望ましい．

定性評価

定性評価とは，数値では表現しにくい性質や印象，現場での実用性などに基づいて評価を行う方法である．食堂混雑の改善策では，「導線が分かりやすい」「交錯が減り安全性が高い」「現場の運用が現実的」「案内が直感的」などの観点が定性評価の対象となる．これらは利用者体験や実装可能性に直結するため，定性評価は数理モデルの評価においても重要な役割を担う．

しかし定性評価は，評価者の主観が入りやすく，評価がぶれやすいという欠点も持つ．この問題を軽減するためには，評価観点を事前に明文化し，複数人の評価を集めたり，インタビューや観察結果などの根拠を添えたりすることが有効である．また，定性評価を単なる感想にしないためには，「なぜその評価となるのか」を因果的に説明し，具体例を挙げることが望ましい．

定性評価は，定量評価と競合するものではなく，互いの欠点を補い合う関係にある．定量評価で効果量を示し，定性評価で現実的な運用可能性やユーザ視点の妥当性を補足することで，総合的に説得力の高い評価となる．

妥当性

妥当性とは，モデルや評価結果が，目的や現実の知見に照らして適切であるといえる度合いである．妥当性は「結果がそれらしく見えるか」という印象ではなく，「仮定や構造が合理的で，目的に対して適切な説明能力を持つか」という観点で考える必要がある．例えば，人流モデルにおいて歩行速度や回避行動が現実と大きく異なる場合，数値的に待ち時間が出力されても，改善案の比較に使うことは危険である．

妥当性には複数の側面がある．内容妥当性（重要要因を取り込めているか），構成妥当性（因果関係が現実の理解と整合しているか），外的妥当性（別条件でも成り立つか）などが代表的である．また，実測データとの比較や現場担当者の知見との照合によって妥当性を高めることができる．

妥当性の確保は，信頼性を支える重要項目である．結論を提示する際には，妥当性を支える根拠（データとの一致，常識的妥当性，極端条件での挙動確認など）を明示し，適用範囲を過度に一般化しないことが必要である．

再現性

再現性とは，同一の条件と手順でモデルを実行した場合に，同等の結果が得られる性質である．研究や開発において再現性は極めて重要であり，再現性が低いと結果の検証ができず，信頼できる結論に到達しにくい．特にシミュレーションでは乱数を用いることが多く，乱数シードを変えると結果が変動するため，再現性を担保する工夫が必要となる．

再現性を確保するためには，モデルの実装，使用したデータ，パラメータ値，実行環境，シミュレーション条件を明確に記録することが重要である．さらに，乱数シードの固定や，実験条件のテンプレート化を行うことで，第三者でも同じ結果を得やすくなる．

再現性が高いことは，単に結果を繰り返せるという意味に留まらない．改善案の効果検証，パラ

メータ変更の影響分析，異常な結果が出た場合の原因追跡など，研究の質を高める基盤となる．

信頼性

信頼性とは，測定やモデル出力が安定しており，偶然や誤差に左右されにくい度合いである．似た条件で繰り返した際に結果が大きくぶれないこと，データの取り方や担当者が変わっても同じ傾向が得られることなどが信頼性の指標となる．例えば待ち時間計測が日によって大きく変動する場合，そのデータのみで結論を出すと危険であり，複数日の観測やサンプル数の確保が必要となる．

シミュレーションにおいても，試行回数が少ないと偶然の要素に左右されやすく信頼性が低下する．そのため，複数回実行した平均値や分散を確認し，安定した傾向が出ているかを評価することが望ましい．また，入力パラメータのわずかな変更で結果が大きく変化する場合は，モデルが不安定である可能性があり，感度分析などによる確認が必要となる．

信頼性の高い結果を示すことで，報告書の提案の説得力が増す．したがって「どの程度安定しているか」「誤差要因は何か」を示し，過度に断定しない形で結論を述べることが重要である．

制約条件

制約条件とは，モデルや施策が満たすべき制限・ルールであり，解の取り得る範囲を規定する条件である．数理モデルでは，変数の範囲，資源制限，空間的制限などが制約条件に相当する．現場課題では，予算，人員，設備，スペース，安全基準，運用ルールなどが代表的な制約条件となる．

例えば混雑緩和として「窓口を増やす」施策が効果的でも，スペースがない，スタッフ確保が困難，設備投資が不可能といった制約があれば実施できない．このように制約条件は，現実的な提案の成立性に直結する．したがって，数理モデルで改善案を検討する際にも，制約条件を明確にし，「実行可能な範囲内で最善を探す」という形にする必要がある．

また，制約条件が曖昧だと，非現実的な結論を導きやすい．よって，制約はできる限り具体化し，どの部分が変更不能で，どの部分が調整可能なのかを整理してモデル化することが望ましい．

境界条件

境界条件とは，モデルが対象とする空間・領域・時間の端において，対象がどのように振る舞うかを定める条件である．人流モデルでは，壁，出入口，通路端，提供口，座席エリア端などが境界に相当し，「壁は通り抜けない」「出口に到達したら退場する」「入口から一定の到着率で入場する」といった形で記述される．

境界条件は，一見すると補助的な設定に見えるが，実際にはシミュレーション結果に強い影響を与えることが多い．例えば入口の位置や入場の仕方（間欠的か連続的か）を変えるだけで，通路混雑の発生場所や行列の形成位置が変わる可能性がある．また，出口を複数設定した場合や，退場時の動線を指定した場合なども結果が変わるため，境界条件は慎重に設定する必要がある．

したがって，境界条件を「モデルの前提の一部」として明確に記述し，その設定理由を示すことが望ましい．境界条件が不明確なモデルは，結果の解釈が困難となり，妥当性や再現性の評価も行いにくくなる．

（※文責: 菊地皓太）

A.1.2　人流 / 空間設計関連用語

人流

人流とは，ある空間（建物内，施設内，街路など）において，人が時間の経過とともに移動する流れや分布の変化を指す概念である．単に「人が歩いている」という意味ではなく，「どこからどこへ，どの経路を通り，どの程度の密度で移動するか」という空間的・時間的構造を含む．例えば食堂では，入口から食事提供へ向かう流れ，座席へ移動する流れ，返却口から出口へ向かう流れなどが複数存在し，それらが同時に発生することで全体の混雑状況が形成される．

人流を扱う際には，人数や移動速度だけでなく，移動方向の偏り，停滞の発生箇所，行列による流れの阻害，動線の交錯なども重要な要素となる．また人流は，施設の配置，通路幅，案内表示，時間帯ごとの来客分布などによって大きく変動する．そのため，人流の把握・モデル化は，混雑緩和策の検討や空間設計の改善における基礎となる．

混雑

混雑とは，ある空間や地点に対して利用者の密度や流量が過剰となり，移動やサービス利用が円滑に行えない状態を指す．混雑は必ずしも「人が多い」ことと同義ではなく，「空間・設備が持つ処理能力や収容能力を超える状態」として捉えることが重要である．食堂の場合，窓口の処理能力（単位時間あたりに提供できる人数）や通路幅，座席数が限られているため，来客が集中すると待ち行列が伸び，滞留が発生し，結果として移動が困難になる．

混雑が発生すると，待ち時間の増加，利用者のストレス，安全性の低下（接触や転倒のリスク増大），スタッフの作業効率低下など，複数の悪影響が生じる．さらに混雑は連鎖的に拡大しやすく，ある地点での停滞が周囲へ波及し，人流全体の流れを悪化させることがある．このため混雑は，単一地点の問題ではなく，施設全体の構造として評価する必要がある．

混雑緩和

混雑緩和とは，混雑の発生頻度や規模を低減し，利用者がより安全かつ快適に施設を利用できる状態へ改善する取り組みである．混雑緩和は単に「人を減らす」ことではなく，人の流れを整理し，ボトルネックを解消し，処理能力を向上させることで，混雑を起こしにくい仕組みを作ることを意味する．食堂では，窓口数や人員配置の最適化，レイアウト変更，動線分離，案内表示の改善，時間差入場の誘導などが混雑緩和策の例として挙げられる．

混雑緩和策を評価する際には，待ち時間や行列長といった定量指標だけでなく，「動線が分かりやすい」「滞留が発生しにくい」「運用が現実的」などの定性的観点も重要となる．また，混雑緩和策にはコストや制約が伴うため，複数案を比較し，現実的に実行可能な範囲で効果を最大化する意思決定が必要となる．

ピーク時間帯

ピーク時間帯とは，施設の利用者が特に集中し，混雑が発生しやすい時間帯を指す．食堂の場合は昼休み開始直後など特定の短時間に利用者が一斉に来訪することが多く，この時間帯に到着率が急上昇するため，待ち行列や滞留が発生しやすい．ピーク時間帯は，平均的な利用状況と異なる特殊条件であり，施設の処理能力の限界が顕在化する時間帯でもある．

ピーク時間帯の混雑は，単純に人数が多いことだけでなく，「短時間への集中」が原因であるこ

とが多い．したがって，ピークを対象にした評価では，到着率の時間変化，窓口の処理能力，座席回転率などを考慮する必要がある．また，ピーク時間帯で成立する混雑緩和策は，通常時間帯にも適用可能な場合が多いため，施設改善の中心的検討対象となる．

滞留

滞留とは，人が本来移動すべき場所で移動できず，ある地点や周辺に留まってしまう状態を指す．滞留は，行列の停止，動線の交錯，目的地の見失い，提供待ち，座席確保の迷いなど，様々な理由で発生し得る．食堂では，食事提供口前での待機，通路上での立ち止まり，返却口周辺での停滞などが典型例である．

滞留が問題となるのは，そこが新たな障害物のように機能し，人流を阻害する点である．滞留が発生すると，その後方の人流が詰まり，通路密度が上昇し，混雑が連鎖的に拡大しやすい．また滞留が常態化すると，安全性低下やスタッフ作業の妨げにもつながる．よって滞留は，混雑の結果としてだけではなく，混雑を引き起こす原因としても評価する必要がある．

動線

動線とは，人が施設内で目的地へ向かって移動する経路や流れのことを指す．動線は個々の利用者の歩行経路だけでなく，利用者全体の流れとして捉えられる．食堂では，「入口→提供口→座席→返却口→出口」といった基本動線が存在し，さらにメニュー選択に応じた分岐動線が重なることで全体の人流が形成される．

動線の良し悪しは，移動のしやすさ，交錯の少なさ，わかりやすさ，安全性などに影響する．動線が不明確である場合，迷い行動や立ち止まりが増え，滞留の原因となる．また動線が狭い通路に集中すると，そこが混雑の起点になりやすい．そのため，混雑緩和策の検討では，動線を可視化し，動線上の問題点を把握することが重要である．

動線設計

動線設計とは，施設内の人の移動経路を意図的に構成し，人流を円滑にするように空間を設計・改善する考え方である．具体的には，通路の配置や幅の調整，提供口の位置，入口・出口の位置，案内表示の設置，仕切りによる流れの誘導などにより，人の流れを整理する．食堂では，提供口への列形成位置を明確にしたり，返却口への流れを提供口の流れと分離したりすることが動線設計の例となる．

動線設計が重要な理由は，人流が自然に分散・整列するように環境を整えることで，運用負荷を増やさずに混雑を抑制できる点にある．また，動線設計は単純な「距離の短さ」だけでなく，「交錯が少ない」「戻り動線がない」「見通しがよい」といった観点も含む．そのため，設計案の評価にはシミュレーションや現場観察を用い，実際に人流がどう形成されるかを検証することが望ましい．

動線交錯

動線交錯とは，異なる目的を持つ利用者の動線が同じ空間で交わり，互いの移動を妨げ合う現象を指す．食堂では，提供口へ向かう人と座席へ向かう人，返却口へ向かう人と出口へ向かう人などが同じ通路上で交差することが多く，ここで動線交錯が発生しやすい．動線交錯は，歩行者同士の回避行動や速度低下を引き起こし，滞留や混雑の原因となる．

交錯が頻発すると，通路全体の処理能力が低下し，行列の進行が遅れ，結果として待ち時間が増加する．また，密度が高い空間での交錯は接触リスクを高め，安全性にも悪影響を及ぼす．そのた

め，混雑緩和の観点では，交錯箇所を特定し，分離・誘導・配置変更によって交錯を減らすことが重要である．

動線遮断

動線遮断とは，ある動線が行列や滞留などによって物理的に塞がれ，他の人流が通行できなくなる状態を指す．食堂では，提供口の行列が通路を占有し，返却口へ向かう通路を塞いでしまう場合などが代表例である．動線遮断が発生すると，本来別目的で移動する利用者の流れまで停止し，混雑が局所から全体へ波及しやすくなる．

動線遮断は，単に通路が狭いことだけでなく，「行列が形成される位置」や「並び方」が適切でない場合にも発生する．例えば列が横に広がる配置になっていると，少人数でも遮断が起こりやすい．したがって，動線遮断を防ぐには，行列形成位置の指定，仕切りの設置，並び方向の制御などを含めた動線設計が必要となる．

ボトルネック

ボトルネックとは，施設内の人流やサービス提供の流れにおいて，処理能力が最も低い箇所となり，全体の流れを制限する要因となる地点や要素を指す．ボトルネックは「瓶の首」の比喩であり，そこが狭いほど流れ全体が滞る．食堂では，提供の処理速度が遅い，通路幅が狭い，入口付近に滞留が集中するなどがボトルネックになり得る．

ボトルネックが存在すると，そこに人が滞留し，行列が形成され，待ち時間が増加する．さらに，ボトルネック周辺の混雑が別の動線を遮断し，混雑が連鎖的に拡大することもある．したがって混雑緩和では，まずボトルネックの位置と原因を特定し，処理能力の増強（増設や人員配置改善）や空間改善（通路拡張や動線分離）によって解消を図ることが重要である．

レイアウト

レイアウトとは，施設内における設備・家具・通路・出入口などの空間配置のことを指す．食堂であれば，提供口の位置，座席配置，返却口の位置，列形成スペース，通路幅などがレイアウトに含まれる．レイアウトは人流形成に直結し，混雑の発生箇所や滞留の起きやすさを決定づける重要要素である．

レイアウトが適切であれば，人流は自然に分散し，交錯が少なく，利用者が迷いにくい環境となる．一方，レイアウトが不適切であると，行列や滞留が通路へはみ出し，動線遮断や交錯が頻発し，混雑が顕著化する．

レイアウト測量

レイアウト測量とは，施設内の構造や配置を把握し，モデル化や設計検討に必要な情報として整理するために，空間寸法や設備配置を計測・記録する作業である．具体的には，通路幅，机や椅子の寸法と配置，壁の位置，入口・出口の位置，列形成可能領域などを測定し，図面化または座標化する．

人流シミュレーションでは，空間の形状が結果に直接影響するため，レイアウト測量の精度はモデルの妥当性や再現性を左右する．例えば通路幅を過大に見積もると混雑が過小評価される可能性がある．したがってレイアウト測量では，実際の施設構造を正確に反映させるとともに，シミュレーション空間へ反映できる形式（座標，グリッド，縮尺図など）で整理することが重要である．

目的地

目的地とは，利用者が移動の最終目標または中間目標として向かう地点を指す．人流モデルでは，個々のエージェントが「どこへ向かうか」を決めることで移動行動が生成されるため，目的地の定義は行動規則の中核となる．食堂であれば，提供口，座席，返却口，出口などが典型的な目的地となる．

目的地の設定は，動線の形成と混雑発生箇所に強く影響する．目的地が集中すると同一方向への流れが生まれ，待ち行列や滞留が発生しやすい．また，目的地間の移動順序をモデルに組み込むことで，現実に近い人流を再現できる．したがって目的地は単なる地点名ではなく，人流モデルにおいて利用者の意図や行動構造を表現する重要概念である．

（※文責: 菊地皓太）

A.1.3　調査 / データ収集関連用語

現地調査

現地調査とは，対象となる場所や状況に実際に赴き，環境の実態や利用状況を直接確認しながら情報を収集する調査である．数理モデリングやシミュレーションにおいては，机上の推測だけでは把握できない要因が多く存在するため，現地調査はモデルの前提設定や妥当性確保の基盤となる．例えば食堂の混雑を扱う場合，来客が集中する時間帯，列が伸びる位置，通路の狭さ，動線の交錯箇所，案内表示の見えやすさなどは，現地での確認によって初めて具体的に把握できる．

現地調査では，空間的情報（レイアウト，通路幅，設備配置）と行動的情報（利用者の動き方，並び方，滞留の発生）を同時に収集できる点が利点である．一方で，観測者の主観や，調査当日の特殊事情（天候，イベント，曜日差）に影響される可能性があるため，調査条件を記録し，可能なら複数回の調査で傾向を確認することが望ましい．また，現地調査で得られた情報は，後の計測・分析と整合するように整理し，モデルへ反映する形に落とし込む必要がある．

計測

計測とは，対象となる量を定めた方法に従って測り，数値として記録する作業である．調査の中でも特に定量的データを得るための手段であり，数理モデルのパラメータ設定や評価指標の算出に不可欠である．食堂の混雑を例にすると，待ち時間，行列長，通過人数，処理時間，利用者の移動速度，滞留人数，ピーク時の入場率などが計測対象になり得る．

計測で重要なのは，測る対象の定義を明確にし，測定方法を統一することである．例えば「待ち時間」を，並び始めから受け取り完了までとするのか，列の最後尾に着いた時刻から窓口到達までとするのかで結果は大きく変わる．また，手作業計測では観測者による誤差が入り得るため，計測手順，測定間隔，サンプル数，測定位置などを明示し，信頼性を確保する必要がある．さらに，計測データは単発では偏りやすいため，複数日の計測や複数回の測定を行い，平均やばらつきを確認することが望ましい．

観察

観察とは，対象の状態や行動を直接見て記録し，特徴や傾向を把握する方法である．計測が「数値を取る」ことに重点があるのに対し，観察は「何がどのように起きているか」を把握することに強みがある．食堂であれば，利用者がどこで迷うか，列が崩れる場面はどこか，動線交錯が頻発す

る地点はどこか，トレーの置き方や会計動作が詰まりを生むか，といった質的情報を得やすい．

観察は，モデルに組み込むべき行動ルールや仮定を構築する際に有効である．例えば「利用者は空いている列に移る傾向がある」「席が空いていないと通路で待つ」「案内が見えにくいと立ち止まる」といった行動パターンは，観察から導かれることが多い．一方で観察は主観が入りやすいので，観察項目を事前に決め，チェックリスト化する，複数人で観察して一致度を確認する，可能なら写真・メモ・時系列記録で根拠を残す，といった工夫が望ましい．また，観察結果は後の定量データと照合しながら解釈することで，説得力が高まる．

実データ

実データとは，実際の現場で計測・記録されたデータ，あるいは運用システムなどから得られる現実のデータのことである．数理モデルやシミュレーションの結果は仮定に基づくため，モデルの妥当性を確認するためには実データとの比較が重要となる．食堂の課題では，特定時間帯の来客数推移，提供処理時間の分布，平均待ち時間，混雑箇所の発生頻度などが実データに該当する．

実データは現実を反映しているという強みがある一方で，欠測や計測誤差，特定条件に偏ったサンプル，例外的な日の影響など，取り扱い上の注意点も多い．また，実データには個人情報やプライバシーに関わる情報が含まれる可能性があるため，収集範囲の適切化や匿名化などの配慮が必要である．実データを用いる際は，取得方法，取得日時，取得条件，データの範囲と限界を明示し，モデル評価に使う際の前提を共有することが望ましい．

データ分析

データ分析とは，収集したデータを整理・加工し，特徴や傾向を抽出して意味づける作業である．単に平均を出すだけでなく，分布，変動，ピーク性，相関，外れ値などを確認し，「何が起きているか」「何が原因と考えられるか」を検討することが目的となる．食堂の混雑であれば，到着率の時間変化を分析してピークの形状を把握したり，窓口別の処理時間の分布を比較してボトルネックを特定したり，待ち時間と行列長の関係を調べて混雑のメカニズムを理解したりすることが該当する．

データ分析は，モデル構築と密接に関係する．分析によって重要要因が見つかれば，モデルに取り入れるべき変数や因果関係の仮定が明確になる．また，モデル出力と実データを比較することで，モデルのパラメータ調整や改善方針を定めることができる．一方で，データ分析はデータ品質に依存するため，欠測処理や外れ値の扱い，サンプル数の不足，測定条件の違いなどを考慮し，分析結果を過度に一般化しないことが重要である．

アンケート

アンケートとは，質問項目を用意し，多数の回答者から意見や状況を収集する調査方法である．混雑問題では，利用者の満足度，待ち時間の体感，混雑箇所の認識，利用頻度，改善要望など，定量データだけでは把握しにくい情報を収集できる．回答を選択式にすれば集計しやすく，傾向を数値として示すことも可能であるため，定性情報と定量情報の橋渡しとして有効である．

アンケートを実施する際には，質問の設計が重要となる．誘導的な質問になると結果が歪むため，中立的な表現を心がける必要がある．また，回答者属性（学年，利用時間帯，利用目的など）を把握できるようにしておくと，条件別の傾向分析が可能になる．一方で，アンケートは回答者の主観に依存し，記憶の誤りや回答の偏りが起き得るため，客観的な計測データや観察結果と併用して解釈することが望ましい．

インタビュー

インタビューとは，対象者に対して対話形式で質問を行い，背景や理由を含めた詳細な情報を得る調査方法である．アンケートが広く浅く情報を集めるのに対し，インタビューは少人数でも深い理解を得やすい．食堂の混雑課題では，利用者へのインタビューにより「なぜその列に並ぶのか」「どこで不便を感じるのか」「混雑回避の工夫をしているか」といった意思決定の理由を把握できる．また，スタッフへのインタビューでは，運用上の制約，ピーク時の対応，提供工程の詰まりやすい点，現場で実施可能な改善策など，外部からは見えにくい重要情報を得られる．

インタビューを有効にするためには，事前に聞きたい仮説や論点を整理し，質問の順序を設計することが望ましい．さらに，回答の内容は主観を含むため，複数の回答者から共通点を抽出したり，観察・計測データと突き合わせて裏付けを取ったりすることが重要である．また，発言の引用や記録を扱う際には，個人が特定されないよう配慮し，研究倫理やプライバシーに十分注意する必要がある．

（※文責: 菊地皓太）

A.1.4　エージェント関連用語

エージェント

エージェントとは，シミュレーションやモデルの中で「自律的に行動する主体」として表現される要素であり，現実世界の人や車両，動物，組織などを模擬する単位である．人流モデルでは，エージェントは主に「歩行者（施設の利用者など）」を指し，各歩行者がそれぞれの目的（提供口へ向かう，座席へ向かう，返却口へ向かう，退場するなど）を持って行動するものとして扱われる．

エージェントは，位置，速度，進行方向といった状態量を持ち，時間の経過とともに状態を更新しながら空間内を移動する．また，エージェントは周囲環境（壁，通路，行列など）や他エージェントの存在を認識し，衝突回避，減速，回避行動などの規則に従って行動する．このように，エージェントは単なる点ではなく，行動ルールを持つ存在として表現されるため，個体差や状況依存の動きをモデルに取り入れやすい．

エージェントを用いる利点は，混雑や滞留などの現象が「個々の行動の積み重ね」として自然に発生することを再現できる点にある．一方で，行動規則の設定が不適切であると，現実とかけ離れた挙動が出る可能性があるため，観察や実データに基づくルール設計とパラメータ調整が重要となる．

マルチエージェント

マルチエージェントとは，複数のエージェントが同一環境内で同時に行動し，相互に影響を与えながらシステム全体が振る舞う状況，またはそのモデルの枠組みを指す．単一のエージェントだけを扱うモデルが「個体の行動再現」に重点を置くのに対し，マルチエージェントでは「多数の個体が同時に存在すること」によって生じる混雑，渋滞，群集の形成といった集団現象の再現が可能となる．

人流の観点では，利用者一人ひとりが同じ空間に存在し，互いに避け合い，追い越し，列に並び，通路を譲り合うなどの行動をとるため，複数エージェントの同時シミュレーションが不可欠である．特に混雑が発生する状況では，他者の存在が移動速度や経路選択に影響し，その影響がさらに別の利用者へ波及するため，システム全体が非線形に変化することが多い．

マルチエージェントモデルでは，人数増加に伴って計算量が増える点や，局所的なルールが全体結果に大きな影響を与える点が特徴となる．そのため，複数の条件での比較や，再現性確保のための試行回数の設定など，モデル運用面での工夫も重要となる．

エージェントベースドモデル

エージェントベースドモデルとは，システム全体の振る舞いを，個々のエージェントの行動規則と相互作用の積み重ねによって表現する数理モデルである．従来のマクロモデルでは，来客数や平均待ち時間のように集団をまとめた量を扱うことが多いが，エージェントベースドモデルでは，個々の主体の状態や意思決定が明示的にモデル化される．そのため，人流のように個人の移動が集団現象を生む問題に対して特に有効である．

食堂の人流を例にすると，各利用者エージェントが「入口から入る」「列に並ぶ」「目的の提供口へ向かう」「受け取り後に座席へ移動する」「食後に返却口へ向かい退場する」といった行動を取り，その途中で他者との回避行動や滞留が生じることで混雑が形成される．このように，現象を上からまとめて計算するのではなく，下から積み上げて再現する点が特徴であり，局所的改善が全体へどう影響するかを検討しやすい．

一方で，エージェントベースドモデルは自由度が高い分，パラメータ設定やルール設計の妥当性が結果に大きく影響する．そのため，現地観察や実データに基づく行動ルールの構築と，結果の妥当性検証が重要となる．また，モデルが複雑になると解釈が難しくなるため，評価指標を用いて結果を整理し，施策比較に活用することが望ましい．

微視的挙動

微視的挙動とは，個々のエージェントが示す細かな行動や動きのことであり，速度変化，方向転換，回避行動，列への合流，立ち止まりなどのレベルの挙動を指す．人流における微視的挙動は，その瞬間の局所環境に強く依存し，周囲の混雑度，壁や障害物の位置，他者の移動方向などによって変化する．

微視的挙動をモデル化する意義は，大規模な混雑や渋滞といった現象が，個々の小さな行動選択の集積から生じるためである．例えば，ある地点で複数人が互いに回避しようとして減速すると，その遅れが後方へ伝播し，行列や滞留が拡大することがある．このような現象は，微視的挙動を表現できるモデルでなければ再現が難しい．

ただし微視的挙動は，モデルのルール設計やパラメータに敏感であり，少しの設定差が全体結果に影響することがある．よって，微視的挙動を扱う場合は，観察結果との整合性を確認し，必要に応じてキャリブレーションを行うことが重要となる．

巨視的現象

巨視的現象とは，多数のエージェントが集団として振る舞うことで現れる，全体規模のパターンや傾向のことである．人流では，混雑の形成，行列の伸長，滞留の発生，流れの分岐や集中，ピーク時間帯の待ち時間増大などが巨視的現象に該当する．これらは個々の行動の結果として現れるが，観測対象としては「全体の状態」として把握される．

巨視的現象は，施設運営上の課題や改善策の効果を評価する際に重要となる．例えば「平均待ち時間がどの程度変わるか」「行列がどこまで伸びるか」「動線交錯が発生する頻度が減るか」などは巨視的な評価であり，意思決定支援に直接利用できる．エージェントモデルは，微視的挙動を規則として与えることで，結果として巨視的現象を再現することが可能となる．

また巨視的現象は，単純な足し算ではなく非線形に現れることがある．例えば来客数が少し増えただけで急激に混雑が発生したり，交錯が一定以上になると通路が遮断されるなどの相転移的変化が生じ得る．そのため，巨視的現象の理解は，混雑緩和策の設計において重要な観点となる．

相互作用

相互作用とは，エージェント同士，あるいはエージェントと環境との間で，互いに影響を及ぼし合う関係を指す．人流では，エージェント同士の距離が近づくと回避行動が生じる，前方の人が減速すると後方の人も減速する，行列への合流が周辺の流れを乱す，といった作用が典型例である．また壁，机，提供口などの障害物も相互作用の対象となり，エージェントは通り抜けを避けたり，沿うように進んだりする．

相互作用は，混雑や滞留といった現象の主要原因であり，個人が単独で移動するときには生じない問題を生み出す．特にピーク時間帯では，エージェント密度が高くなるため相互作用が強まり，移動効率が急激に低下する．このことは，混雑が単に人数の増加ではなく，密度増大による相互作用の増加によって非線形に悪化する理由を説明する．

モデル化の観点では，相互作用はルールや力（Social Force Model の反発力など）として表現される．相互作用の強さや範囲の設定は結果に大きく影響するため，現実データや観察に基づいたパラメータ調整が必要となる．

行動特性

行動特性とは，エージェントがどのような傾向や規則に従って行動するかを表す性質であり，目的地選択，速度，経路選択，回避行動，待ち行列への並び方，滞留の起こしやすさなどに反映される．現実の利用者は，皆が同じ行動を取るわけではなく，急いでいる人，慎重に歩く人，列に並ぶ際に空いている場所を探す人など，個体差が存在する．このような差をモデルに取り入れることで，人流シミュレーションの現実性が高まる．

行動特性は，環境条件によって変化する場合もある．例えば混雑時には速度が落ちる，交錯が多いと回避距離を大きく取る，案内が不十分だと迷い行動が増える，といった状況依存性がある．したがって，行動特性を設計する際には，単純な固定値ではなく，密度や距離などの状態に応じて変化するルールとして表すことがある．

行動特性は巨視的現象に強い影響を及ぼす．例えば「列に並ぶ際に 2 列形成を好む」という特性を入れるだけで，行列の形状が変わり，通路遮断の発生率が変化する可能性がある．そのため，行動特性の設定には根拠が必要であり，観察・インタビュー・実データに基づいて仮定を置くことが望ましい．

（※文責: 菊地皓太）

A.1.5 Social Force Model 関連用語

Social Force Model

Social Force Model（SFM）とは，歩行者の移動を「社会的な力（Social Force）」の作用として表現する，人流シミュレーションに用いられる代表的なモデルである．ここでいう力は物理的な押す力そのものというより，「目的地へ向かいたい」「他人や障害物を避けたい」といった意思決定を，数式上の力として扱う概念的なものである．モデルの基本的な考え方は，歩行者（エージェン

ト）の加速度が複数の力の合成によって決まり，その結果として速度や位置が時間とともに更新されるというものである．

SFM では，目的方向へ進むための希望推進力，他者や壁などから距離を保つための反発力，接触時の物理的相互作用に相当する項などを組み合わせて歩行挙動を表す．この枠組みにより，単純な最短経路移動では表現しにくい，混雑時の速度低下，回避運動，流れの形成，渋滞の発生といった群集特有の現象を自然に再現できる．

食堂の混雑問題では，狭い通路，待ち行列形成，動線交錯といった要因が存在するため，SFM を用いることで「人が密集したときにどのように動きにくくなるか」を含めた現実的な人流再現が可能になる．一方で，SFM はパラメータ設定に結果が強く依存するため，現地データに基づいた調整や妥当性検証が重要となる．

歩行者挙動モデル

歩行者挙動モデルとは，人が歩行する際の行動や移動の仕方を，数理的または計算機上で表現するためのモデルの総称である．歩行者は道路や施設内で，自身の目的地へ向かいながら，他者や障害物を避けたり，状況に応じて速度や経路を変えたりするため，その挙動を表すモデルが必要となる．

歩行者挙動モデルには多様な種類があり，Social Force Model のように連続空間で力を用いて表すモデル，格子状空間で移動規則を定めるセルオートマトン型モデル，エージェントベースドモデルとして意思決定ルールを明示するモデルなどが代表例である．それぞれ表現できる現象や計算量が異なるため，目的や状況に応じて適切なモデルを選択する必要がある．

本課題では混雑や動線交錯が重要となるため，個人同士の距離関係を反映できる歩行者挙動モデルが有効である．特にピーク時の高密度状況では，個々の歩行者の回避行動が群集全体の流れに影響を与えるため，微視的挙動を再現できるモデルの採用が望ましい．

希望推進力

希望推進力とは，Social Force Model において，歩行者が「望ましい速度と方向」で移動しようとする意思を表す力である．歩行者は目的地へ向かう際に，ある希望速度（快適に歩ける速度）と希望方向（目的地方向）を持つと考えられ，現在の速度との差を埋めるように加速しようとする．これを数式では推進力として表現し，歩行者が目標の移動状態へ近づくように作用させる．

希望推進力は，歩行者が混雑していないときには比較的強く働き，歩行者は希望速度に近い速度で移動できる．一方で周囲が混雑している場合，反発力や回避の影響によって希望推進力が十分に発揮できず，結果として速度低下や滞留が生じる．このように，希望推進力は「進みたい」という行動原理を表す項であり，混雑状態の変化を再現する上で中心的な役割を持つ．

また，希望推進力の設定（希望速度の分布，反応時間など）は，混雑の程度や流れの形成に影響する．例えば希望速度が高い設定だと追い越しや回避が増え，交錯が増加する可能性がある．そのため，現地観察に基づくパラメータ設定が重要となる．

反発力

反発力とは，Social Force Model において，歩行者が他者や障害物と距離を保とうとする作用を表す力である．歩行者は周囲の人や壁に近づきすぎると不快感や危険を感じるため，一定距離を確保するように移動方向や速度を調整する．この心理的・行動的な回避傾向を，数式上の反発力として表すことで，衝突を避けるような自然な動きを再現する．

反発力は，歩行者間距離が短くなるほど強く働くように設定されることが多く，結果として密度の高い状況では速度低下や渋滞が発生しやすくなる．また壁や机などの障害物に対しても反発力を設定することで，通り抜けや接触を防ぎつつ，壁沿いを移動するような挙動も表現できる．

反発力の強さや作用範囲のパラメータは，人流再現に大きく影響する．反発力が弱すぎると人同士が不自然に接近し，強すぎると過剰な回避で流れが乱れる．そのため，実データとの比較やキャリブレーションを通じて適切な設定を行うことが望ましい．

障害物回避

障害物回避とは，歩行者が壁，テーブルなどの物理的障害物を避けながら移動する行動である．人流シミュレーションにおいて障害物回避を表現しない場合，エージェントが壁を通り抜けたり，机を横切ったりする非現実的挙動が発生するため，モデルの妥当性が大きく損なわれる．

Social Force Model では，障害物からの反発力を設定することで障害物回避を実現することが多い．障害物に近づくほど反発力が強くなり，歩行者は自然に障害物から離れる方向へ力を受ける．さらに，狭い通路では障害物回避が複数方向から働くため，流れが細くなり，ボトルネックが発生しやすくなる．このような現象を再現するためにも，障害物回避の適切なモデル化は重要となる．

障害物回避は，人流の混雑評価だけでなく，空間設計にも直結する．例えば，机の配置を変えると回避行動の頻度が変化し，動線交錯や滞留の発生状況が変わるため，レイアウト改善案の評価に役立つ．

衝突回避

衝突回避とは，歩行者同士が接触や衝突を避けるために，進行方向の変更，速度低下，一時停止などの行動を取ることを指す．混雑した空間では歩行者同士の距離が短くなるため，衝突回避行動が頻繁に発生し，結果として移動効率が低下する．食堂のように動線交錯が起こる環境では，衝突回避が連鎖して滞留や渋滞を引き起こす要因となる．

Social Force Model では，衝突回避は主に歩行者間の反発力によって表現される．すなわち，他者との距離が縮まるほど反発力が強まり，進行方向が変化したり減速したりすることで衝突が回避される．また，高密度環境では反発力が強く働くため，速度が落ち，前方の遅れが後方へ伝播することで群集全体の渋滞が形成される．

衝突回避の適切な表現は，モデルの安全性評価にも関係する．衝突リスクが高い箇所を特定できれば，動線設計の改善や案内の配置変更など，現実的な混雑緩和策の検討につながる．

群集流動

群集流動とは，多数の歩行者が集団として移動する際に生じる，全体的な流れや動きのパターンを指す．個々の歩行者が自律的に動いていても，密度が高くなると集団全体として一定の方向へ流れる，通路に沿って列状の流れが形成される，交錯地点で渦状の滞留が起きるなど，個体の単純な和では説明できない現象が現れる．このような集団としての動きを，群集流動として扱う．

群集流動の特徴は，個人行動が他者に影響し，その影響がさらに波及して全体の流れが決まる点にある．例えば前方の速度低下が後方へ伝播して渋滞が形成される現象は，車両交通だけでなく歩行者群集にも見られる．Social Force Model は，個体間の反発力や希望推進力の合成により，群集流動を自発的に生成できるため，群集現象の再現に適したモデルとして利用される．

群集流動を把握することは，混雑緩和策の検討において重要である．例えば流動の方向を分離する，交錯を回避する，流れの幅を確保するなど，空間設計への具体的な指針を得ることができる．

高密度空間

高密度空間とは，一定の面積あたりに存在する歩行者数が多く，歩行者同士の距離が短くなる空間を指す．食堂のピーク時間帯では，提供口前や狭い通路，返却口付近などで高密度空間が形成されやすい．高密度になると個人の自由な移動が制限され，衝突回避や減速が頻発し，流れが詰まりやすくなる．

高密度空間は，混雑の評価において重要な概念である．密度が一定値を超えると，移動速度が急激に低下し，滞留が発生しやすくなるなど，状態が非線形に変化することがある．さらに安全性の観点からも，高密度環境では接触リスクや転倒リスクが増大するため，混雑緩和策における優先検討対象となる．

Social Force Model では，高密度空間では反発力が強く働き，結果として速度低下や群集流動の変化が生じる．このため，高密度状況における挙動再現は，モデルの妥当性を左右する重要事項となる．

流入密度

流入密度とは，単位時間あたりに空間へ流入する人の量を表す概念であり，到着率や入場流量と関連する指標である．食堂の混雑では，入口から入ってくる利用者数が時間帯によって変化し，特にピーク時間帯では流入密度が高くなる．流入密度が高いほど，処理能力や通路容量を超過しやすく，待ち行列形成，滞留，動線遮断といった混雑現象が発生しやすい．

流入密度は，混雑発生の主要原因であるため，モデル化や改善策検討では重要な入力条件となる．例えば，流入密度が同じでも窓口数が増えれば混雑が緩和される可能性があり，逆に窓口数が同じでも流入密度を時間分散できればピーク混雑が低減される．このように流入密度は，システムの需要側を表す指標として，供給側の処理能力と対比しながら扱う必要がある．

また，流入密度を設定する際には実データに基づくことが望ましい．推定が不適切だと混雑を過大評価または過小評価するため，計測結果や既存統計から妥当な値を採用することが重要である．

キャリブレーション

キャリブレーションとは，モデルが現実の挙動を適切に再現するように，モデル内のパラメータを調整し整合性を高める作業である．Social Force Model では，希望速度，反発力の強さ，反応時間，障害物回避の範囲など複数のパラメータが存在し，それらの設定によって移動速度や混雑の発生状況が大きく変化する．したがって，モデルの妥当性を確保するにはキャリブレーションが不可欠となる．

キャリブレーションは，実測データとの比較に基づいて行われる．例えば，実際の平均歩行速度，行列長の変化，待ち時間分布，混雑箇所の位置などを基準にし，モデルの出力がこれらに近づくようにパラメータを調整する．また，単に平均値が一致するだけでなく，時間変化の傾向やピーク時の挙動なども整合しているかを確認する必要がある．

キャリブレーションを行うことで，モデルは施策比較の道具として信頼性が高まる．ただし，特定条件に合わせすぎると他条件での再現性が低下する可能性があるため，複数条件での確認や，適用範囲の明示が重要となる．

（※文責: 菊地皓太）

A.1.6　待ち行列理論関連用語

待ち行列

待ち行列とは，サービスを受けたい利用者が，サービス提供側の処理を待つために形成する列，およびその状態を指す．現実の場面では，レジ待ち，食事提供の受け取り待ち，などが代表例である．待ち行列は単なる「列」ではなく，到着する利用者の流入と，サービス処理による流出が時間的に釣り合わないときに生じる動的な現象である．

食堂のようなサービスシステムでは，ピーク時間帯に到着者が集中すると待ち行列が伸び，待ち時間が増加する．また待ち行列が通路にはみ出すと動線遮断や滞留を引き起こし，人流全体の混雑悪化につながる．そのため，待ち行列は混雑評価の中心的要素であり，待ち行列の長さや待ち時間を指標として改善策を検討することが多い．

モデル化の観点では，待ち行列は「利用者が一定の規則で到着し，窓口で順に処理される」という構造を持つシステムとして扱うことができ，これが待ち行列理論の対象となる．

待ち行列理論

待ち行列理論とは，サービスを受けるために発生する待ち行列の性質を，確率論や数理モデルを用いて解析する理論体系である．待ち時間や行列長がどの程度になるか，窓口数を増やすとどの程度改善されるか，混雑がどの条件で急激に悪化するかといった問題を定量的に扱える．待ち行列理論は，飲食店の提供口やレジに限らず，通信ネットワーク，交通，病院受付，コールセンター，計算機システムなど幅広い分野で用いられている．

待ち行列理論では，利用者の到着の仕方（到着率），サービス処理の時間，窓口数，待ち順序，システムの容量などを仮定し，その下で平均待ち時間や平均行列長を導出する．特に M/M/1 型や M/M/c 型など，確率分布を仮定した基本モデルは解析が可能であり，混雑の理解や施策検討に役立つ．

ただし待ち行列理論は，多くの場合「到着がランダム」「サービス時間が指数分布」「定常状態が成立する」などの仮定に基づく．現実の食堂ではピーク集中などがあり，必ずしも理論の仮定通りではないため，理論モデルの適用範囲を意識し，必要に応じてシミュレーションや実データと併用することが重要である．

待ち行列長

待ち行列長とは，待ち行列に並んでいる利用者数を表す量であり，混雑の程度を直接示す基本指標である．待ち行列長には，瞬間的な時刻における列の人数，一定時間の平均行列長，最大行列長などの表現がある．食堂では，提供口前の列が何人まで伸びるかは，通路遮断や滞留の発生と密接に関係するため重要な指標となる．

待ち行列長は，到着率がサービス処理能力を上回ると増加し，下回ると減少する．つまり，到着率とサービス能力の差が行列長の増減を決定する．このため，待ち行列長を観測・モデル化することで，どこがボトルネックになっているか，窓口を増やすべきか，列形成位置を変更すべきかといった改善策を検討しやすくなる．

また，行列長は「待っている人数の多さ」を視覚的に示すため，利用者のストレスや行動（列を避けて別メニューへ向かう，利用を諦める）にも影響する．そのため，単なる内部指標ではなく，施設運用上の重要な評価項目である．

平均待ち時間

平均待ち時間とは，利用者がサービスを受けるまでに待つ時間の平均値を表す指標である．混雑評価において最も代表的な指標の一つであり，利用者満足度に直結する．食堂では，列に並び始めてから料理を受け取るまでの時間，あるいは窓口到達までの時間など，定義により待ち時間の意味が変わるため，測定・評価に用いる際は定義を明確にする必要がある．

平均待ち時間は，単に平均値を示すだけでなく，ピーク時と非ピーク時で大きく変化することが多い．また平均が短くても，一部の利用者が極端に長く待つ場合があるため，最大待ち時間や分布も併せて評価することが望ましい．特に食堂の運用改善では，「平均待ち時間を短縮しつつ，極端な待ちを発生させない」ことが重要となる．

待ち行列理論では，平均待ち時間は到着率，サービス時間，窓口数などによって決まり，特に混雑度が高いと非線形に増大することが知られている．つまり少しの需要増加が大きな待ち時間増大につながる場合があるため，ピーク対応では重要な評価指標となる．

到着率

到着率とは，単位時間あたりにサービスシステムへ到着する利用者数の平均的な割合を表す量であり，待ち行列理論の基本パラメータである．通常，λで表され，例えば「1 分あたり 5 人が到着する」といった形で定義される．食堂においては，入口から入ってくる利用者数，または特定窓口へ並び始める利用者数が到着率となる．

到着率は時間によって変動することが多く，特にピーク時間帯では急激に上昇する．このため，ピーク時の到着率を正しく把握しなければ，混雑を過小評価し，現実的でない結論を導く危険がある．また到着率は行列形成を決定づける主要因であり，到着率がサービス能力を超えると行列が増え続ける状態になり得る．

実務上は，到着率を測定することで需要の強さを把握でき，窓口数増設，運用変更，時間分散策などの検討に利用できる．モデル化では，到着率を一定と仮定する場合もあるが，現実性を高めるために時間依存の到着率を導入することもある．

サービス時間

サービス時間とは，1 人の利用者に対してサービス提供が完了するまでに必要な時間を指す．待ち行列理論では，サービス時間の平均や分布がシステムの処理能力を決めるため重要な要素である．食堂では，注文受付から提供完了まで，盛り付けや会計に要する時間，受け渡し動作に必要な時間などがサービス時間に含まれる．

サービス時間が長いほど，単位時間あたりに処理できる人数が減少し，混雑が発生しやすくなる．またサービス時間は一定ではなく，注文内容の違い，スタッフの熟練度，ピーク時の作業負荷などにより変動する．そのため，平均だけでなくばらつきも混雑に影響し得る．例えばサービス時間のばらつきが大きいと，処理の遅い利用者が列全体を停滞させ，待ち時間が増加しやすい．

改善策としては，作業工程の短縮，スタッフ配置の最適化，支払い方式の変更などによりサービス時間を短縮できる可能性がある．したがってサービス時間の計測・分析は，ボトルネック特定と改善案設計において重要となる．

窓口数

窓口数とは，同時にサービス提供を行えるサーバ（提供口，レジ，担当者など）の数を指す．待

ち行列理論では，窓口数はシステムの処理能力を直接決定するパラメータであり，混雑緩和策の代表的な操作変数である．食堂では，提供口の数や，会計担当者の人数などが窓口数に相当する．

窓口数を増やすと，同時処理能力が増加し，平均待ち時間や行列長が短縮されやすい．ただし窓口数を増やすには，人員確保，設備追加，スペース確保などの制約条件が伴う．また窓口を増やしても，列形成スペースが不足して通路遮断が起こったり，別の工程がボトルネックになったりする場合がある．

したがって窓口数は重要な改善要素ではあるが，単純に増やせばよいとは限らない．レイアウトや動線設計と合わせて総合的に検討し，実行可能性と効果のバランスを取ることが必要となる．

システム混雑度

システム混雑度とは，待ち行列システムがどの程度混んでいるかを表す指標であり，通常は「利用率」または「トラフィック強度」として定義される．一般に，到着率 λ と窓口数 c，サービス率 μ（1 単位時間あたりの処理能力）から求められる．ここで ρ が 1 に近いほど混雑が激しく，1 を超えると行列が増え続けて安定しない状態になる．

混雑度は，混雑の発生可能性を簡潔に評価できる利点がある．例えば到着率が少し増えるだけで ρ が 1 に近づく状況では，待ち時間が急激に増大するリスクが高い．この非線形性はピーク混雑の説明に有効であり，「少しの需要増加が大きな混雑悪化を生む」現象を示す根拠となる．

ただし，混雑度は単純化された指標であり，現実のサービス時間分布や時間変動をすべて含むものではない．そのため，混雑度は目安として用い，実データやシミュレーション結果と併用して解釈することが望ましい．

M/M/1 型モデル

M/M/1 型モデルとは，待ち行列理論における代表的な基本モデルであり，到着過程がポアソン過程（Markovian 到着），サービス時間が指数分布（Markovian サービス），窓口数が 1 つであるシステムを表す．この名称の M/M/1 は，それぞれ到着分布，サービス分布，窓口数を示している．

M/M/1 型は解析解が知られており，平均待ち時間や平均行列長などを数式で求めることができるため，混雑の基本的性質を理解する上で有用である．例えば混雑度 ρ が 1 に近づくと待ち時間が急増することは，M/M/1 の解析式から明確に示される．

ただし，実際の食堂では窓口が複数存在することが多く，また到着がピークに偏るため，M/M/1 の仮定と異なる場合が多い．そのため，M/M/1 は現象理解や概算に有効な一方，詳細な再現には M/M/c や非定常モデル，またはシミュレーションが必要となる．

M/M/c 型モデル

M/M/c 型モデルとは，到着がポアソン過程，サービス時間が指数分布であり，窓口が c 個存在する待ち行列モデルである．M/M/1 を一般化したモデルであり，複数窓口による同時処理を表現できるため，現実の窓口サービスに近い形で扱える．食堂では，提供口が複数ある場合や，レジ担当が複数いる場合などが M/M/c で近似される．

M/M/c では窓口数が増えるほど処理能力が増し，平均待ち時間や行列長が減少しやすい．一方，窓口数が増えても到着率が極端に高ければ混雑は発生するため，到着率と窓口数のバランスが重要となる．また窓口数が多いほどサービスが分散され，混雑が緩和されるが，運用コストも増大するため，費用対効果の観点から最適な窓口数を検討することが多い．

現実適用の際には，「窓口が完全に同等である」という仮定が成り立たない場合がある点に注

意が必要である．例えば窓口ごとにメニューが異なるとサービス時間や到着率が異なり，単純な M/M/c では表現しにくい．その場合は窓口別モデル化やエージェントシミュレーションが有効となる．

優先度

優先度とは，複数の利用者が待っている際に，誰を先にサービスするかを決める規則や順位の概念である．一般的な待ち行列では先着順（FIFO）が多いが，優先度を導入すると，特定の利用者が優先的に処理される．例えば，急病患者を優先する病院受付，緊急度の高い通信パケットを優先するネットワーク制御などが典型例である．

食堂の運用では明示的な優先度は少ないが，例えば「一部メニュー専用窓口」を設けることは，結果としてメニュー別の優先度を生む場合がある．また返却口の流れを優先して通路を確保する設計をすると，安全性のための優先度設計とみなすこともできる．

優先度は，待ち時間の公平性に影響する．優先対象があると非優先側の待ちが増える可能性があるため，運用上の目的（効率，安全，満足度）と公平性のバランスを考慮して設計する必要がある．

バッチ処理

バッチ処理とは，複数の対象を個別に順次処理するのではなく，一定数をまとめて一括で処理する方式を指す．待ち行列の文脈では，サービスが「1 人ずつ」ではなく「まとめて」提供される場合にバッチ処理として扱われる．例えば，料理をまとめて盛り付けする，トレーを一定数まとめて洗浄する，あるいは列がある程度たまってから処理を進めるような運用が該当し得る．

バッチ処理は，単位処理あたりの効率を高められる場合がある．例えば同じ作業を連続して行うことで段取り時間が減り，結果として処理能力が向上することがある．一方で，バッチ処理は「まとめるための待ち」を発生させるため，利用者側の待ち時間が増える可能性もある．つまり効率向上と待ち増加のトレードオフが存在する．

食堂においても，特定のメニューがまとめて作られる場合はバッチ的な提供が起きている可能性がある．その影響を考える際には，サービス時間が一定ではなく波状になる点を意識し，必要なら非定常な処理モデルとして扱うことが望ましい．

非定常モデル

非定常モデルとは，到着率やサービス率が時間によって変化し，システムが定常状態（統計的に一定の状態）に落ち着かない状況を扱う待ち行列モデルである．一般的な待ち行列理論の基本モデル（M/M/1 や M/M/c）は，パラメータが一定で長期的に安定した状態を仮定することが多い．しかし現実の施設運用では，昼休み開始直後に来客が集中するなど，到着率が大きく時間変動するため，定常仮定が成り立たない場合が多い．

食堂の混雑問題は典型的に非定常であり，ピーク時間帯に到着率が急上昇し，その後減少するという時間依存性を持つ．この場合，ピークの短時間に行列が急増し，ピーク後も行列処理の遅れが残るため，平均的な到着率で評価すると混雑を過小評価する危険がある．したがって，ピーク特性を正確に扱うには，非定常モデルや時間依存到着率を含むシミュレーションが有効となる．

非定常モデルは解析が難しい場合が多いが，現実性が高いという利点がある．ピーク混雑が中心課題となる場合には，非定常性を考慮したモデル化が重要となる．

（※文責: 菊地皓太）

A.1.7　シミュレータ開発関連用語

シミュレータ

シミュレータとは，現実の現象やシステムをモデル化し，計算機上で再現・実行することで，挙動の観察や評価を行うためのソフトウェアまたは仕組みである．実際の施設で改善策を試すことは，コストや時間，安全性の観点から難しい場合が多い．そのため，シミュレータを用いて仮想的に条件を変え，結果を比較することで，現実施策の意思決定支援に活用できる．

本プロジェクトでは，食堂内の人流や行列形成を計算機上で再現し，レイアウトや運用条件を変更したときに混雑がどう変化するかを調べる目的でシミュレータを開発する．シミュレータには，モデル（数式や行動ルール）の実装，入力データの読み込み，パラメータ設定，結果計算，出力・可視化などの機能が含まれる．また，結果の信頼性を高めるためには，再現性確保，条件管理，実データとの整合性確認なども重要である．

仮想環境

仮想環境とは，現実の空間やシステムを計算機上に再現した環境であり，シミュレーションが実行される舞台に相当する．人流シミュレーションでは，食堂の壁，通路，窓口，座席配置などを仮想的な空間として構築し，その内部でエージェントが移動することで人流を再現する．仮想環境は単なる背景ではなく，移動可能領域と障害物領域，出入口位置などを含み，エージェント挙動に直接影響する要素である．

仮想環境の精度は，モデル結果の妥当性を左右する．例えば通路幅が実際より広いと混雑が過小評価され，窓口位置や列形成スペースの再現が不正確だと動線交錯の発生箇所が変化する可能性がある．したがって仮想環境を構築する際は，レイアウト測量などに基づき，実空間の主要特徴を適切に反映し，モデルが扱う範囲と限界を明確にする必要がある．

可視化

可視化とは，シミュレーションや計算の結果を，図，グラフ，アニメーションなどの視覚的表現に変換し，理解しやすい形で提示することである．数値の表だけでは把握しにくい現象も，動線の流れや滞留の発生箇所を可視化することで直感的に理解できる．特に人流シミュレーションでは，エージェントの移動軌跡，混雑密度の変化，行列の形成過程などをアニメーションとして示すと，混雑の原因や改善案の効果を説明しやすい．

可視化は研究の説明手段であると同時に，モデルの検証手段にもなる．例えば，エージェントが壁を貫通していないか，回避行動が自然か，行列形成が現実と整合しているかなどは，可視化によって早期に確認できる．また，結果を共有する際にも，関係者が直感的に理解できるため，意思決定支援として有効である．

反復的評価

反復的評価とは，モデルやシミュレータを一度作成して終わりにするのではなく，実行結果を評価し，問題点を修正し，より良いモデルへ改善するサイクルを繰り返すことを指す．人流シミュレーションでは，初期モデルの段階で現実とずれた挙動が生じることが多く，パラメータ調整，行動ルール変更，仮想環境修正などを繰り返すことで妥当性を高めていく．

反復的評価の利点は，モデルの精度を徐々に高められるだけでなく，「どの要因が結果に影響するか」を段階的に把握できる点にある．例えば，窓口処理時間を調整すると待ち時間が大きく変わる，動線設計を変えると交錯が減るなど，改善の方向性が明確になりやすい．ただし反復を有効にするには，条件や変更点を記録し，どの修正が結果を改善したかを追跡できるようにすることが重要である．

比較検証

比較検証とは，複数の条件や施策案を同一基準で比較し，どの案が目的に対して有利かを評価するプロセスである．シミュレータの強みは，現実環境では試しにくい複数案を安全かつ低コストで比較できる点にある．例えば，レイアウトを変更した場合，待ち行列の形成方式を変更した場合，動線分離を行った場合などを同じ到着率条件で実行し，平均待ち時間や行列長を比較することで，改善効果を定量的に示せる．

比較検証で重要なのは，「比較条件を揃えること」である．到着率，シミュレーション時間，評価指標の定義が異なると，公平な比較にならない．また乱数要素がある場合は複数回試行して平均値とばらつきを比較し，偶然の影響を排除することが望ましい．このように比較検証は，施策の意思決定支援に直結する重要な工程である．

パラメータ

パラメータとは，モデルやシミュレーション内で固定値として扱われる設定値であり，現象の性質を決める数値的要素である．人流モデルでは，希望速度，反発力の強さ，到着率，サービス時間，回避距離などがパラメータとして設定される．パラメータの値は結果に大きく影響するため，適切に設定されていないと混雑が過大・過小に評価される恐れがある．

パラメータは，実データに基づいて設定することが望ましい．例えば平均歩行速度や処理時間を計測して反映することで現実性が高まる．またパラメータの調整作業はキャリブレーションに相当し，妥当性確保のための重要工程である．さらに，パラメータには不確実性があるため，感度分析を行い，どのパラメータが結果に強く影響するかを確認することも有効である．

プロトタイプ

プロトタイプとは，最終成果物の完成版を作る前に，主要機能や概念を試すために作成する試作品である．シミュレータ開発では，いきなり全機能を作り込むのではなく，まず簡易的なプロトタイプを作成し，動作確認やモデル構造の妥当性を検討することが効率的である．例えば，単純なレイアウトでエージェントが目的地へ到達できるか，行列形成が再現できるかを試し，問題点を発見して改善する．

プロトタイプの利点は，早期に課題を発見できる点にある．実装上の不具合，想定外の挙動，パラメータ設計の難しさなどを早期に把握できるため，最終的な開発リスクを低減できる．またプロトタイプは関係者への説明にも使いやすく，改善案の議論を具体化する材料にもなる．

要件定義

要件定義とは，開発するシステムが「何を実現すべきか」を明確にし，必要な機能，性能，制約，入力，出力などを整理する工程である．シミュレータ開発においては，目的が混雑緩和策の比較なのか，人流可視化なのか，待ち時間推定なのかによって必要機能が変わるため，要件定義を行うことで開発の方向性が明確になる．

要件定義では，例えば「レイアウト変更を入力として扱える」「到着率を時間依存で与えられる」「結果として待ち時間と行列長を出力できる」「アニメーション表示が可能」といった要件を設定する．また，利用可能なデータ形式，計算時間の上限，実行環境なども要件に含めることがある．要件定義が不十分だと，開発途中で目的がぶれたり，不必要な機能を作り込んでしまったりするため，初期段階での整理が重要である．

基本設計

基本設計とは，要件定義で定めた要件を満たすために，システム全体の構造を決める設計工程である．具体的には，システムをどの機能モジュールに分割するか，データの流れをどうするか，入力と出力の形式をどう定義するか，主要アルゴリズムをどう配置するかなどを定める．

シミュレータでは，仮想環境管理，エージェント管理，力計算，更新処理，可視化，データ入出力などの役割を分離し，保守しやすい構造にすることが望ましい．基本設計が適切であれば，後の改良や機能追加が容易になり，反復的評価をスムーズに進められる．逆に基本設計が曖昧だと，実装が場当たり的になり，バグ修正や拡張が困難になる．

詳細設計

詳細設計とは，基本設計で決めた構造をもとに，各モジュール内部の具体的な処理内容を決める工程である．例えば，エージェントが持つ状態変数（位置，速度，目的地など），力の計算式，衝突判定方法，更新順序，データ形式，クラス構造などを具体化する．

詳細設計では，アルゴリズムの正確性だけでなく，実装可能性や処理効率も考慮する必要がある．人流シミュレーションではエージェント数が増えると計算量が増大するため，距離計算の効率化やデータ構造の工夫が求められる場合がある．また，詳細設計を明確にしておくことで，複数人での開発や，後からの改修が容易になる．

UI

UI（User Interface）とは，利用者がシステムを操作し，結果を確認するための画面や操作方法のことである．シミュレータの UI には，パラメータ入力欄，開始停止ボタン，レイアウト選択，結果表示，グラフ表示などが含まれる．UI が整備されていると，専門知識が少ない利用者でも条件を変えて実行でき，施策比較が容易になる．

人流シミュレータでは，UI を通じて「到着率を変える」「窓口数を変える」「レイアウト案を切り替える」などの操作が可能になると，検証効率が大きく向上する．また，結果が UI 上で視覚的に提示されることで，混雑の原因や改善効果が直感的に理解できる．ただし UI の開発は時間やコストもかかるため，必要十分な範囲に絞り，プロトタイプ段階では簡易 UI を用いることも多い．

CSV ファイル

CSV ファイル（Comma Separated Values）とは，データをカンマ区切りで記述したテキスト形式のファイルであり，表形式データを扱う代表的な形式である．多くのソフトウェアで扱いやすく，Excel などで編集できるため，シミュレータの入力データや出力データの保存によく用いられる．

人流シミュレーションでは，時間帯別来客数，到着スケジュール，窓口処理時間の記録，シミュレーション結果（待ち時間，行列長，滞留人数など）を CSV として保存することで，再分析や可視化が容易になる．また CSV 形式であれば，他の分析ツール（Python，R など）に読み込んで統

計処理やグラフ作成を行うことも簡単である．

ただし CSV は単純な形式であるため，列の意味や単位がファイルだけでは分かりにくい場合がある．そのため，列名の明示，単位の統一，欠測値の扱いなどを整理し，データの再利用性と再現性を高めることが重要である．

（※文責: 菊地皓太）

A.1.8　数値計算関連用語

微分方程式

微分方程式とは，未知の関数とその導関数との関係を式として表した方程式であり，「状態がどのように変化するか」を数学的に記述するための基本形式である．多くの現象は「変化の速さが現在の状態によって決まる」という性質を持ち，それを表現すると自然に微分方程式となる．例えば運動の法則では，速度や加速度が力によって決まり，位置が時間とともに変化するため，位置・速度・加速度の関係は微分方程式で記述される．

シミュレータ開発の観点では，微分方程式は「連続時間で変化するモデル」を表す際に用いられる．本プロジェクトで扱う Social Force Model でも，エージェントの速度変化（加速度）が希望推進力や反発力の合成で決まるため，位置・速度の更新は微分方程式に基づく形になる．ただし実際の開発では，微分方程式を解析的に解くことは難しいため，離散化して数値的に近似解を求め，ステップ毎に状態更新する形でシミュレーションが行われる．

このため微分方程式は，モデル構造を決める理論的基盤であると同時に，シミュレータ実装における更新ルールの出発点となる．

常微分方程式

常微分方程式（ODE）とは，未知関数が 1 つの独立変数だけに依存する微分方程式である．つまり，状態量が時間とともに変化する現象を表現する際に用いられ，最も基本的な動的モデルの形として幅広く利用される．例えば位置 x(t)，速度 v(t) の時間変化は ODE で表される．

人流シミュレーションでは，各エージェントの位置や速度が時間の関数として変化するため，エージェントの運動方程式は ODE として表現できる．Social Force Model においても，加速度が力の合成として与えられ，速度と位置が時間積分で更新されるため，実質的に ODE を数値計算していることになる．

ODE は解析的に解ける場合もあるが，多くは非線形で複雑なため数値解法が必要となる．シミュレータではオイラー法などの数値積分法によって，時間ステップごとに状態を更新し，連続時間の近似を行う．その際，時間ステップの設定や誤差の扱いが結果に大きく影響するため注意が必要である．

偏微分方程式

偏微分方程式（PDE）とは，未知関数が複数の独立変数に依存し，偏微分が含まれる微分方程式である．代表例として，熱の拡散方程式，波動方程式，流体のナビエ・ストークス方程式などが知られている．PDE は空間全体での分布の変化を扱うため，連続体の現象や密度の時間変化を表すのに適している．

人流の扱いにおいても，個々の人を追跡するミクロモデルとは別に，「人の密度分布が空間でどのように変化するか」を連続量として扱うマクロモデルが存在し，その場合に PDE が用いられることがある．例えば，混雑密度が時間と空間でどのように変化するかを表すモデルは PDE で表現され得る．

ただし PDE を数値的に解くには，空間方向の離散化も必要となり，計算コストが大きい．本プロジェクトのようにエージェントを用いた人流シミュレーションでは主に ODE が中心となるが，密度マップ生成や混雑度評価の理論背景として PDE 的な考え方が関連する場合もある．

離散化

離散化とは，連続的に変化する量（時間，空間，速度など）を，計算機で扱えるように有限個の点や区間に区切って近似する操作である．微分方程式に基づくモデルは本来連続時間で定義されるが，計算機は連続無限の情報を扱えないため，離散化によって有限の計算手順へ落とし込む必要がある．

シミュレータ開発では，特に時間離散化が重要となる．連続時間の変化を「Δｔごとに状態更新する」形にし，時刻の点で位置や速度を更新していく．また PDE のような空間を含むモデルでは，空間もグリッドに区切って離散化する．

離散化は便利である一方，近似誤差を生む．区切り幅が粗いと現象を正しく再現できず，細かくすると精度は上がるが計算量が増える．したがって離散化では，計算負荷と精度のバランスを取る設計が必要となる．

時間ステップ

時間ステップとは，シミュレーションで時間を離散化したときの 1 回の更新に対応する時間間隔Δｔのことである．シミュレータでは，時間ステップごとにエージェントの速度や位置を更新し，連続時間の近似として時間発展を再現する．

時間ステップの値は，結果の精度と計算コストを直接左右する．時間ステップが大きすぎると，エージェントが壁を突き抜ける，回避行動が不自然になる，行列形成が乱れるなどの数値的不安定が生じやすい．一方で小さすぎると更新回数が増えて計算が重くなり，実行時間が長くなる．

そのため，時間ステップはモデルの特性（最大速度や力の強さなど）に応じて適切に選ぶ必要がある．一般に，人流のように衝突回避を扱う場合は，短い時間ステップの方が挙動を安定して再現しやすい．また，時間ステップは再現性確保の観点でも重要であり，使用値を明示することが望ましい．

収束判定

収束判定とは，反復計算や数値解法において，解が十分に安定し「これ以上計算しても大きく変わらない」と判断できる状態に到達したかを確認する手続きである．数値計算では，解析解が得られない場合に近似解を反復的に求めることが多く，その計算をどこで止めるかを決めるために収束判定が必要となる．

具体的には，反復前後の値の差がある許容誤差以下になった，残差が一定値以下になった，更新量が十分小さくなった，などの基準で判定を行う．例えば非線形方程式をニュートン法で解く場合や，反復法で線形方程式を解く場合に収束判定が用いられる．

シミュレータ開発では，ODE の時間積分そのものは収束判定が不要な場合もあるが，パラメータ調整，最適化，平衡状態探索，あるいは数値計算アルゴリズムを組み込む際には収束判定が重要

となる．収束判定が甘いと精度不足，厳しすぎると計算時間過大となるため，許容誤差設定は実用性と精度を両立させる必要がある．

数値誤差

数値誤差とは，数値計算において理想的な真値と計算結果の間に生じるずれのことであり，シミュレーション結果の精度に影響する重要要因である．数値誤差は大きく分けて，離散化による誤差（打ち切り誤差），浮動小数点計算に伴う丸め誤差，アルゴリズムの反復誤差などから生じる．

例えばオイラー法で ODE を解く場合，時間ステップ幅 Δt に比例する打ち切り誤差が発生し，時間ステップを大きくすると誤差が増大する．また多数回更新を行うと丸め誤差が累積し，長時間シミュレーションで結果がずれていく可能性もある．さらに非線形系では誤差が増幅されやすく，小さな誤差が大きな挙動差につながることがある．

数値誤差を抑えるためには，時間ステップの適切化，高次精度の数値解法の使用，試行条件の統一などが有効である．また，誤差の影響を確認するために，ステップ幅を変えて結果が安定しているかを調べる検証も重要となる．

非線形方程式

非線形方程式とは，未知数が線形の形で現れない方程式の総称であり，未知数の積，べき乗，指数関数，三角関数などが含まれる場合に非線形となる．非線形方程式は，一般に解析的に解くことが難しく，数値計算によって近似解を求めることが多い．

非線形性は，複雑な現象を生む原因となり得る．例えば，入力条件が少し変わるだけで出力が大きく変化する，複数の解が存在する，急激な状態変化が起こるなどの特徴が現れることがある．人流のような問題でも，混雑が一定値を超えると急に渋滞が発生するといった非線形的振る舞いが生じる場合がある．

Social Force Model でも，反発力が距離に対して指数関数的に変化するなど，非線形性を含む場合が多い．この結果，シミュレーションは非線形ダイナミクスを持ち，数値誤差や初期条件に敏感になる可能性がある．

初期値依存性

初期値依存性とは，システムの初期状態（初期値）が少し違うだけで，時間発展後の結果が大きく変わる性質を指す．特に非線形系や混雑が発生しやすい系では，初期条件のわずかな違いが，行列の形成位置，滞留の発生タイミング，混雑度のピークなどに影響し得る．

人流シミュレーションでは，エージェントの初期位置，初期速度，入場タイミング，目的地の割り当てなどが初期条件となる．例えば，数人の入場タイミングが少しずれただけで，動線交錯の発生頻度が変化し，結果として混雑状況が大きく変わる場合がある．このため，初期値依存性が強い場合，単一試行の結果だけで結論を出すのは危険である．

初期値依存性への対処としては，複数回試行して平均的傾向を評価する，乱数シードを固定して比較条件を揃える，ばらつきも含めて結果を提示する，といった方法が有効である．また初期値依存性を理解することは，「混雑が偶然によって悪化しやすい構造」を見つける手がかりにもなり，混雑緩和策の設計にも役立つ．

（※文責: 菊地皓太）

A.1.8.1　非線形方程式の解法

はさみうち法

はさみうち法とは，方程式の解が含まれていると分かっている区間を用意し，その区間を徐々に狭めることで解に近づく数値解法の総称である．例えば関数の値が区間の左端と右端で異なる符号を持つ場合，区間内に少なくとも 1 つ解が存在することが保証される．この性質を利用して，解を含む区間を保ったまま繰り返し区間を縮めることで，解の近似値を得る．

はさみうち法の利点は，初期区間の取り方が適切であれば，基本的に安定して解へ近づく点にある．収束の信頼性が高いため，発散しやすい手法の補助としても有効である．一方で収束速度は速くない場合が多く，高精度を得るには反復回数が多くなることがある．二分法は，このはさみうち法の代表例である．

二分法

二分法とは，はさみうち法の中でも最も基本的な方法であり，解を含む区間を毎回ちょうど半分に分割して区間を更新する手法である．まず解を含むことが保証された区間を用意し，その中点を計算する．次に，中点での関数値の符号を調べ，どちら側に解が存在するかを判定し，その側の半区間を次の探索区間として採用する．これを繰り返すことで，区間幅が着実に半分ずつ縮小し，解の近似精度が向上する．

二分法は非常に単純でありながら，「必ず区間が狭まる」ため安定性が高い．また導関数を必要とせず，関数値の計算だけで進められる点も利点である．一方で収束は比較的遅く，精度を上げるためには反復回数が必要となる．実務では，まず二分法で解の位置を大まかに定め，その後ニュートン法などの高速法で精密化する使い方がよく行われる．

ニュートン法

ニュートン法とは，方程式の解を高速に求める反復法であり，関数を局所的に直線近似して次の解候補を更新していく方法である．イメージとしては，現在の推定値における接線を引き，その接線が横軸と交わる点を次の推定値とする手続きを繰り返す．解に近い初期値が与えられる場合，少ない反復回数で急速に高精度へ到達しやすいことが特徴である．

ニュートン法の利点は収束が速い点であり，条件が良いと誤差が急激に小さくなる．一方で，初期値が悪いと発散したり，別の解に収束したりする場合がある．また関数の傾き（導関数）が極端に小さい場合，更新が不安定になりやすい．したがってニュートン法は万能ではなく，初期値を二分法などで安全に作ってから適用する，もしくは更新幅を制限するなどの工夫と併用されることが多い．

（※文責: 菊地皓太）

A.1.8.2　数値積分

台形法

台形法とは，定積分の値を数値的に近似する方法の一つであり，積分区間を細かい区間に分割し，各区間の面積を台形として近似して足し合わせる手法である．各小区間では，関数の曲線を直線で近似して面積を求めるため，図形的には「台形の面積を合計する」ことに相当する．

台形法は実装が簡単で，離散データ（測定値など）が与えられている場合にも適用しやすい．そ

のため，シミュレーション結果のグラフデータから累積量を求める場面などでも利用される．一方で，関数の曲がりが大きい場合には誤差が増えやすく，精度を上げるには分割数を増やす必要がある．

シンプソン法

シンプソン法とは，定積分をより高精度に近似する数値積分法であり，台形法が直線近似を用いるのに対し，シンプソン法は小区間内の関数を 2 次曲線（放物線）で近似して面積を求める．このため，同じ分割数で比較すると台形法より精度が高くなりやすい．

シンプソン法は，関数が滑らかである場合に特に有効であり，少ない分割数でも誤差を抑えやすい利点がある．一方で，区間分割数の扱いに条件がある場合や，関数が急激に変動する場合には注意が必要である．実装としては台形法より少し複雑になるが，高精度な積分値を得たい場合に広く用いられる．

（※文責: 菊地皓太）

A.1.8.3　データ近似と補間

最小二乗法

最小二乗法とは，観測データに対して最もよく当てはまる近似式（直線や曲線）を求める方法であり，「誤差の二乗和が最小になる」ように係数を決定する手法である．例えば散布図データに対して回帰直線を求める場合に最も代表的に用いられる．

最小二乗法が重要なのは，データには測定誤差やばらつきが含まれることが多く，単に点を全て通る曲線を作るよりも，全体として最も妥当な傾向を捉えることが目的となるからである．シミュレーションでは，実測データから到着率やサービス時間の平均傾向を推定したり，パラメータの関係を近似してモデル化したりする際に利用される．

残差平方和

残差平方和とは，近似式やモデルがデータをどれだけ説明できているかを示す指標であり，各データ点における「観測値とモデル値の差」を二乗して全て足し合わせた量である．残差平方和が小さいほど，モデルがデータに良く一致していると考えられる．

最小二乗法は，この残差平方和が最小になるように近似式の係数を求める方法である．ただし残差平方和が小さければ必ず良いモデルとは限らず，複雑な式にすると過剰にデータに合わせすぎる（過学習）ことも起こり得る．そのため，残差平方和はモデルの当てはまりを評価する重要指標ではあるが，モデルの簡潔さや解釈性なども合わせて考慮する必要がある．

ニュートンの補間法

ニュートンの補間法とは，与えられた複数の点を全て通る補間多項式を構成する方法であり，点を追加したときに計算を効率的に拡張できる形で表現できることが特徴である．補間とは，既知のデータ点の間の値を推定する操作であり，シミュレーションで得られた離散データから連続的な曲線を復元する場面などで重要となる．

ニュートンの補間法は，差分を用いた段階的な構成が可能であり，新しい点を追加するたびに最初から作り直さなくてもよいという利点がある．一方で，高次数の多項式補間は不安定になりやすく，点の数が増えると振動が発生する場合があるため，利用には注意が必要である．

スプライン補間

スプライン補間とは，区間ごとに低次数の多項式を用い，それらを滑らかに接続して補間曲線を作る方法である．全区間を 1 つの高次多項式で表す補間に比べ，局所的に安定しやすく，不自然な振動が起きにくいという利点がある．

スプライン補間では，各データ点を通りつつ，点の境界で曲線が滑らかにつながるように条件を課す．これにより，全体として自然な曲線が得られやすい．実データの時間変化（到着率の時間依存性など）を滑らかに復元したい場合や，グラフを美しく補間したい場合に有効である．

ランゲ現象

ランゲ現象とは，多項式補間を行った際に，区間の端付近で曲線が大きく振動し，不自然な値を示す現象である．特に等間隔に点を取って高次数の補間多項式を作ると，端付近で振動が増大しやすいことが知られている．

ランゲ現象は，「点を増やして補間を高精度にしようとしたのに，かえって不安定になる」という直感に反する問題を生む．そのため実務では，多項式補間を安易に高次数化することは避け，スプライン補間など局所的で安定な補間手法が選ばれることが多い．

（※文責: 菊地皓太）

A.1.8.4　線型方程式

緩和法

緩和法とは，連立方程式などを解く際に，解を一度で求めるのではなく，推定値を徐々に改善して解へ近づける反復的手法の総称である．ここでの「緩和」は更新の強さを調整する意味を持ち，更新量をそのまま適用するのではなく，適度に弱めたり強めたりすることで収束を安定化させる．

緩和法の考え方は，解の計算を安定に進めることにあり，特に大規模な方程式系や数値解法でよく用いられる．適切な緩和係数を選ぶと収束が速くなることがある一方，不適切だと収束が遅くなったり発散したりするため，設定に注意が必要である．

掃き出し法

掃き出し法とは，連立一次方程式を解くための直接解法であり，係数行列を段階的に変形して解を求める方法である．一般にガウス消去法の一形態として理解でき，行基本変形を用いて未知数を順に消去し，最終的に解を導出する．

掃き出し法の利点は，反復を必要とせず，原理的には有限回の操作で解に到達できる点にある．ただし計算量が大きく，大規模問題では計算負荷やメモリ使用量が増えやすい．また数値誤差や丸め誤差の影響を受けやすい場合もあるため，計算機上では安定化の工夫が必要となる．

ガウス・ザイデル法

ガウス・ザイデル法とは，連立一次方程式を反復的に解く代表的手法であり，各未知数を順番に更新しながら解へ近づける方法である．ヤコビ法と似ているが，ガウス・ザイデル法は更新した値を直ちに次の計算に利用するため，収束が速くなる傾向がある．

この方法は，大規模行列に対して計算負荷を抑えながら近似解を得たい場合に有効である．特に

係数行列が対角優位である場合，収束しやすいことが知られている．一方で，収束性は行列の性質に依存し，条件が悪い場合は収束しないこともあるため注意が必要である．

反復法

反復法とは，解を一度で求めず，初期推定値から出発して繰り返し更新を行い，解へ近づけていく数値解法の総称である．ガウス・ザイデル法やニュートン法などは反復法に含まれる．反復法は大規模問題に適用しやすく，メモリ消費が比較的小さいという利点がある．

ただし反復法では，収束判定が必要であり，更新を何回行えば十分かを決める必要がある．また，初期値やパラメータによって収束が遅くなったり，発散する可能性もある．そのため，収束性の条件や誤差評価を併せて考えることが重要である．

直接法

直接法とは，反復によって近似解を求めるのではなく，有限回の計算手順によって解を求める数値解法の総称である．代表例が掃き出し法（ガウス消去法）であり，理論上は誤差がなければ正確に解を得られる．

直接法は，反復回数や収束判定を考えなくてよい点が利点である．一方，大規模問題では計算量とメモリが膨大になることがある．また計算機では丸め誤差が避けられないため，実際には数値誤差の影響を受ける．そのため，大規模計算では反復法が選択されることも多い．

対角優位行列

対角優位行列とは，連立一次方程式の係数行列において，各行の対角成分の絶対値が，同じ行の他の成分の絶対値の和以上となる性質を持つ行列である．この性質は，反復法の収束性と強く関係している．特にガウス・ザイデル法などの反復法では，係数行列が対角優位であると収束しやすいことが知られている．

直感的には，対角成分が十分大きいと，各未知数が自分自身の項によって強く支配され，他の未知数の影響が相対的に小さくなるため，更新が安定する．そのため数値計算では，反復法を適用可能かどうかの判定材料として対角優位性が重要視される．

（※文責: 菊地皓太）

A.1.8.5　常微分方程式の数値解法

テイラー法

テイラー法とは，関数をある点の近傍で展開した近似式（テイラー展開）を用いて，微分方程式の解を数値的に更新する方法である．状態の次の値を「現在の値と，変化の情報の組み合わせ」として求めるという考え方であり，微分係数の情報を高次まで利用するほど精度が上がる．

ただし実装上は，高次の導関数を求める必要があり，モデルが複雑になると導関数計算が困難になる．そのためテイラー法は理論的基盤として重要である一方，実務では導関数を直接求めなくても高精度が得られるルンゲ＝クッタ法などが広く用いられる．

オイラー法

オイラー法とは，常微分方程式を数値的に解く最も基本的な方法であり，「現在の変化率が次の短時間でも一定である」と近似して次の値を計算する．実装が非常に容易で，シミュレーション開

発の初期段階やプロトタイプでよく利用される．

一方，オイラー法は精度が高くなく，時間ステップが大きいと誤差が蓄積しやすい．また力が急変する場合や衝突回避を含むモデルでは不安定になりやすい．そのため，オイラー法を用いる場合は時間ステップを十分小さくする，もしくはより高精度な解法に移行するなどの対応が必要となる．

ルンゲ＝クッタ法

ルンゲ＝クッタ法とは，常微分方程式を高精度に解くための代表的数値積分法であり，1 ステップの中で複数回変化率を評価し，それらを組み合わせて次の値を決定する．一般に 4 次のルンゲ＝クッタ法が広く利用され，精度と計算負荷のバランスが良い．

ルンゲ＝クッタ法の利点は，オイラー法より高精度で，時間ステップを多少大きくしても安定しやすい点にある．特に人流シミュレーションのように，力の変化が複雑で数値誤差が蓄積しやすい問題では有効である．一方で，1 ステップあたりの計算回数が増えるため計算負荷が上がる．そのため，精度要求と計算時間を踏まえて解法を選択することが必要である．

（※文責: 菊地皓太）

A.1.8.6　空間系の数値解法

差分法

差分法とは，微分方程式に含まれる微分を「近傍点の差」を用いて近似し，連続問題を離散的な計算問題に置き換える手法である．時間や空間を格子点に分割し，隣接点の値の差から微分を表現することで，更新式を作る．

差分法は実装が比較的単純で，計算機上で扱いやすい．ただし格子の刻み幅の取り方によって誤差が変化し，安定性条件を満たさないと発散することがある．そのため，刻み幅の設計と数値誤差評価が重要となる．

FDTD 法

FDTD 法（Finite-Difference Time-Domain 法）とは，時間領域で差分法を用いて電磁場などの波動現象を数値的に解く手法である．時間と空間を格子に分割し，各時刻で場の量を更新していくことで，波の伝播を直接シミュレーションできる．

FDTD 法は，複雑な形状の媒質や境界条件を含む問題にも適用できる一方，計算格子が細かいほど計算負荷が急増する．そのため，精度と計算量のバランスを取ることが重要である．

有限要素法

有限要素法（FEM）とは，対象領域を多数の小領域（要素）に分割し，各要素内で近似関数を用いて解を表現する数値解法である．差分法が格子点の差を中心に構成されるのに対し，有限要素法は不規則形状を扱いやすく，複雑な境界を持つ問題に強い．

有限要素法では，領域全体を分割して局所的な近似を行い，それらを全体として整合するように組み合わせる．構造解析，熱伝導，電磁場解析など，多くの工学分野で標準的に利用される手法であり，現実的形状を扱える点が大きな利点である．

ガラーキン法

ガラーキン法とは，微分方程式を近似解空間に射影して解を求める手法であり，有限要素法の理論的基盤の一つとして位置付けられる．近似解を複数の基底関数の線形結合として表し，誤差が特定の条件（重み関数との直交条件）を満たすように係数を決める．

この方法は，厳密解を直接求めるのではなく，「誤差の成分を抑える」考え方で近似解を得る．そのため，境界条件や複雑な領域に対しても安定した近似が得られやすい．

コロケーション法

コロケーション法とは，微分方程式の近似解を仮定し，いくつかの選んだ点（コロケーション点）において方程式が厳密に満たされるように係数を決定する手法である．すなわち連続的な条件を全区間で満たす代わりに，代表点で一致させることで計算問題へ落とし込む．

コロケーション法は実装が比較的簡単で，高精度近似が得られる場合もある．一方で，点の選び方によって精度や安定性が変化し，局所的に誤差が集中することもある．

部分領域法

部分領域法とは，誤差を全領域で制御するのではなく，領域を複数の部分領域に分け，それぞれの領域内で方程式の条件が満たされるように近似解を決める手法である．全体の平均的な誤差を抑えるという発想であり，ガラーキン法などとも関連する．

部分領域法は，問題の性質が場所によって異なる場合や，特定の領域で精度を確保したい場合に有効である．ただし設計には工夫が必要であり，領域の分割方法が結果に影響することがある．

アンペールの法則

アンペールの法則とは，電流が磁場を生み出すことを表す電磁気学の基本法則である．直感的には，「電流が流れるとその周囲に磁場が発生する」という関係を定量的に示す．この法則は電磁場解析の基礎となり，回路理論や電磁波解析に広く応用される．

数値計算の文脈では，電磁場の支配方程式（マクスウェル方程式系）を扱う際に重要となる．FDTD 法などの電磁場シミュレーションは，アンペールの法則を含む方程式を離散化し，時間発展として計算することで場の変化を求める．

ファラデーの法則

ファラデーの法則とは，時間的に変化する磁場が電場を生み出すことを表す電磁気学の基本法則である．直感的には，「磁場が変化すると誘導電流が生じる」という電磁誘導の原理を定量的に示す．発電機，トランス，電磁誘導加熱などの基礎となる重要な法則である．

数値計算の観点では，ファラデーの法則もアンペールの法則と並んでマクスウェル方程式を構成し，電磁波の伝播を支配する．FDTD 法はこれらの関係を時間領域で計算する代表的手法であり，ファラデーの法則に相当する更新式を繰り返して電場・磁場を時間発展させることで解析を行う．

（※文責: 菊地皓太）

A.2　提案されたテーマ案

本プロジェクトにおいて，テーマ決めのフェーズで提案されたテーマ案を以下に示す．

人流・施設レイアウト最適化
・集客施設におけるテーブル・椅子配置の検討
例：受験会場，レストランなどでの席配置と動線の設計
・小売店舗での顧客動線シミュレーション
店内を細かく分割し，顧客同士の位置関係や滞留を分析
・計画・非計画購買者を考慮した店舗内人流シミュレーション
視力・商品属性などにより立ち止まる確率をモデル化
・未来大食堂の人流モデル（待ち行列モデル含む）
行列待ち時間や混雑発生メカニズムの説明
・駅における人流推定（AI ＋数理モデル）
入場→受け取り→着席→返却→退場の行動と回避行動の推定
通路ごとの人通り計測，歩行軌跡の計測

※人流シミュレーションを用いた混雑原因の説明技術
ミクロダイナミクス分析：案内板を見る／食事目的など，選択行動に着目
参考：富士通 Technical Review 2020 論文
※ SFM（Social Force Model）を用いたパラメータ測定手法
目的地への引力，人同士の反発力など相互作用を含む
参考：DEIM2017 資料

社会シミュレーション・都市シミュレーション
・社会シミュレーション設計ノウハウの整理（モデルパターン化）
参考：SICE（計測自動制御学会）論文
・都市衰退過程の空き家発生・集積と空地の商業影響
マルチエージェント都市シミュレーション（人口・土地利用の変化）
参考：J-STAGE 論文
・都市の人口増加の予測
人口推移・都市成長モデルの構築

感染症・リスク推定
・都市間の感染症変動のモデル化
パンデミック時の集まり・移動傾向から感染リスクの高いエリアを特定

環境・省エネルギー制御
・人流と外気温を用いた換気量・空調出力制御による省エネ効果検証
人流データ＋翌日最高気温予測を用いた省エネ実証

スケジューリング・最適化（勤務・シフト）
・シフトスケジューリング問題（一般）
週あたり希望時間，時間帯責任者など制約の下で最適シフト作成
参考：MSI（スケジューリング解説）
・コールセンターのスケジューリング

勤務時間・人員配置制約を満たすシフト作成
・相性を考慮したナーススケジューリング
看護師間の相性評価を制約／目的関数に反映
・能力を考慮した勤務シフトの遺伝的アルゴリズム解法
世代進化による適応度改善の検証
・数理計画モデルによる熟練者エージェントへの接近
熟練／未熟練による処理時間・開始時間の変化に注目

予測・推定（データ分析系）
・天気予測
・株価・為替の変動予測
・曜日別来客数・売上予測（観光地の中小飲食店）
天候・学内イベント等による変動要因も考慮

可視化・教育工学
・視覚情報の可視化がオンライン学習の理解度把握に与える影響
参考：J-STAGE（教育工学系）論文

その他
・演奏：生音（耳）と録音の音の違いの分析
・錯覚による交通渋滞
・人工意識（意識モデル）
・魚の移動モデル（函館の漁港での活用可能性）
・区間内に解が 2 つ以上あるときの処理（数値計算／解の選択）

（※文責: 菊地皓太）

A.3　発表で寄せられた意見

中間発表技術コメント

ポジティブな意見
・声が聞こえやすく，目的が何かがわかりやすかった
・動画の再生などがスムーズだと良いと思った
・シミュレーション動画を用いている点が良かった.
・面白かったです
・スライドに画像や動画が多用されており分かりやすい
・なんとか動画再生できててよかったです
・シミュレーションの動画がありわかりやすい
・わかりやすい説明で，とてもよく練習されているように思いました.
・図を指して説明したり，聞き手の方をしっかり見ていてよかった

・ハキハキ説明してたので，とても聞きやすい説明だと思いました．
・とても良かった
・身近な内容を実例として出していて分かりやすかった
・計測や観察から気付いた問題点をしっかりと提示して具体的でわかりやすかった．またプロトタイプでの解説がわかりやすく今後の活動をどんなことしてくのかが見やすかった

アドバイス
・早口で聞づらい
・途中早口で何を言ってるのか聞き取れませんでした
・早口で聞き取りにくい部分があった．これは緊張由来だと思うので，一回深呼吸してから始めてみたほうが良いと思う．
・各セクションにおける簡潔な説明を最初に入れていただけると，理解しやすいと思います
・もう少し大きい声だとより聞こえやすかったです
・機材切り替えで手間取っていたのはなぜ？ PC をみずに発表できると良かったですね（ほとんど聴衆と目線が合ってなかった）

ポジティブとアドバイスが混在してる意見
・スライドが見やすくとても良かったと思います．ただ，シミュレーションは最後まで見せて欲しかった思います
・スラスラ発表していて良かったと思います！声量をもう少し大きくすると遠くにいる人も聞こえるので意識してみると良いと思います．
・呼びかけや，気になる話題を使った発表の流れ，スライドの密度などはわかりやすく良かった．すこしテンポ感が速く，声量がもう少し大きいと聞きやすくさらに分かりやすくなると思いました．
・台本を見ないで発表できていてよかったと思います．声が若干 Make Brain に負けていました．ワンオペっぽかったのが気になりました．
・やってきたことについて，伝わりました．目的と，そのために何をするかが少し不明瞭だった
・冒頭で，取り組むテーマについて，共感する人に挙手を求めたり，具体的な場面を話題にしたりと，理解を得る工夫がなされていた．スライドでは抽象化した図で示し，図中の要素を口頭で説明することで，着目している問題点を理解しやすくなっていた．投影機器の不良に焦らず対処していたのはよかったが，機器が復帰するまで繋ぎの説明（例．次のページで何を話す予定か）があるとよりよいと感じた．
・エージェントの反発の「反発」が何かもう少し教えて欲しかったです
・とても面白い取り組みだと思いました．待行列のモデルとエージェントに基づく歩行のモデルはどのように接続されるのか興味深いです．今後楽しみにしています．それからベクトルじゃなくて太字で表現してほしいです．高校生ではないのでね．

中間発表内容コメント

ポジティブな意見
・難しいことを図やデータを用いて解説してくれているためわかりやすかったです．

・良いと思った
・理解しやすい研究テーマで良かった.
・アプローチがしっかりできていてすごいと思いました.
・よかったです
・食堂という身近なテーマに対し，数理モデリングを利用して擬似的に食堂内の様子を再現し，解決方法を探るというやり方が良いと思いました.
・未来大の食堂の混雑という身近なを取り上げていて良いと思う
・目標設定がおもしろい
・シュミレーションを組み込んでいて分かりやすかった
・実際に自分たちでデータを集めて，それを元に数式を立て，どのような方法を用いれば原因の解決につながるのかまとめられていて良かった.
・とても良かった
・問題分析や，技術解説がよくできていてわかりやすかったです.
・プロトタイプもできていて，いいペースで進んでいると思います．数理モデル作成後の食堂を見てみたいです
・既にシミュレータの実装にまで着手されていて，今後が楽しみです．曜日の違いなどもシミュレータで表せれば面白いと思います.

アドバイス
・数理モデルの説明は，畑が異なると数式の羅列にしか見えないことがありがちだと思う．今回の発表については，「どこでどんなモデルを用いたか」は説明の筋が通っていてかなりわかりやすかった．一方で，「希望推進力とは？」のような説明は，モデルの説明 1 ページでまとめるのも重要である．それに加えて，学食で人がどう行動するのか？（進みたい方に進む，他者とぶつからないよう避ける）のような説明は，口頭だけでなくスライドにも補足があると良いと思う.
・食堂の混みは，人流モデルを使って解決できるんですか？どうしても 2 限終わりは混むんじゃないかと思うんですけど.
・食堂シミュレーションとしてとても良くできていると思うが，これと待ち時間の短縮に繋がる未来が見えなかった
・スライドに入れている数式や図の各項目がどのような意味なのかまったく注釈がない.
・人の動きがリアルだった．人流の改善で，本当に混雑が解消するかの疑問はある.
・なにをシミュレーションする？ シミュレータの元データはどうやって計測？ シミュレーションの当てはまり具合はどうやって判断？ 最終的にどうしたい？ モデル改良って何をするの？（より現状に合ったシミュレーターの作成？）

ポジティブな意見とアドバイス
・モデル式の構築やシミュレーションの実装具合は，現実の状況に近いもので素晴らしいと思います．混雑状況を解消する具体的な手段が決まるとより良いプロジェクトになると思います.
・全体的なプロジェクトの流れがわかりやすかったです！ Social Force Model の「各エージェントの反発力」について，どのような気持ち（原因）で反発力が発生しているのか説明するとよりモデルが伝わりやすくなると思いました！
・曜日ごとの混み具合についてデータにまとめていて，金曜 2 限終わりが混むことがわかったが，

なぜ混むのかの予測がされていなかったのでその情報が欲しかった
・シュミレーターの動きは面白かったです．でも混雑をどう解消していくかイメージしづらかった
・モデルの中身についてはもう少し詳しい説明が必要だと感じた
・背景からはじまり，だんだん深い内容に入っていたので聞き取りやすかった．「食堂の混雑を軽減する」という目標において，後半に紹介していたモデリングのシュミレーターはあくまで手段の 1 つだと思うので，そこからどう結論を出すかが大事になると思う

成果発表技術コメント

ポジティブな意見
・動画が組み込まれててわかりやすかったです
・スライドよくまとまっていて，わかりやすかったです．
・次のスライドに進む時人が発表する人と分かれているため，非常にスムーズでした
・モデリングがかなり精巧で面白かった

アドバイス
・グラフが見にくい　特に軸のラベルをもう少し減らして大きくするなどの工夫があるといい
・少し声が小さくて聞こえにくい時があったのでもう少し声を大きくするといいと思います．
・声が小さく聞き取れなかった
・画面を見ていて，声が小さいため発表の内容が聞き取れませんでした
・声が聞こえにくかった
・もう少し聞こえやすい声量での発表をお願いしたいです．また，聴衆側に向けて話すのが発表の仕方として適当だろうと思いました．
・終始，顔が聴衆ではなく画面のほうを向いているので，声が非常に通りづらくなっている．時間と時刻は使い分けたほうがいい．

ポジティブとアドバイスが混在してる意見
・スライドがとても見やすいです．もう少し大きな声で発表できるとよかったです．
・発表の声を聞き取ることはできなかったが，スライドに図表が効果的に使われていたので，内容を理解することはできた．
・スライドは見やすいものの，声の小ささが気になった．

成果発表内容コメント

ポジティブな意見
・仮説実験考察と順序立てて発表がされていて良かったです．
・意義のあるテーマだと思います．
・テーマぎめは，身近で介入可能なところで上手だと思いました．
・食堂を利用する側なのでとても面白かったです．
・計画取りと身近に改善ができるテーマ，それに向けての具体的解決法案がしっかりしており，内容はとても良いと感じた．

・データやアンケートの結果に基づいて仮設を立て，検証している部分がよい．
・声をはって聞き取りやすい声で発表していた．
・面白いシミュレーションができていたと思います．
・わかりやすいスライドと説明だった

アドバイス
・移動区間を離散的に区切って，ネットワーク科学的な中心性の議論をしても面白いかなと思いました．
・今後の展望に関して，数理モデリングとの関連や詳細な説明がなかったため，混雑の分析に数理モデリングを用いた目的がわからなかった．開店時と 2 限終了時に混みやすいことはひと月程度食堂を利用すれば経験的に知ることができるし，列を増やすことで長大化を避けられることも一般的な経験則に基づいている．数理モデリングを用いることで，経験知ではわからない知見を得られた部分はどこなのか，どういった洞察を得ることができると考えて数理モデリングを行ったのかが知りたい．

ポジティブとアドバイスが混在してる意見
・身近な課題解決について数理的な方法でアプローチしている点がよいと思いました．二列化とレイアウト変更というソリューションをどう思いついたのか知りたかったです

（※文責: 鈴木創太）

A.4　シミュレータ実装コードの解説

ここでは，シミュレータ開発におけるコードについて，本文中で扱いきれなかった補足事項や詳細な解説をまとめる．

（※文責: 菊地皓太）

A.4.1　インポートと実行環境設定

コード冒頭では「本シミュレータが依存するライブラリ」と「表示環境（日本語表示）に関する設定」を行い，以降の処理が正常に動作する基盤を整えている．本シミュレータは，Matplotlib を用いた可視化・UI 操作を中心に，NumPy によるベクトル計算，Pandas による入退場データ（CSV）の読み込み，および Python 標準ライブラリによる乱数・日時処理を組み合わせて実装している．

（※文責: 菊地皓太）

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.patches as patches
4 from matplotlib.animation import FuncAnimation
```

```
from matplotlib.widgets import CheckButtons, Button, RadioButtons,
↪  RectangleSelector
from matplotlib.lines import Line2D
import random
import pandas as pd
from datetime import datetime, timedelta
import copy
from itertools import combinations
from matplotlib.animation import FFMpegWriter
import layout
```

NumPy：本シミュレータでは，エージェントの位置・速度・力を 2 次元ベクトルとして扱う．そのため，ベクトルの加算・減算・正規化・距離計算などを効率的に実行できる NumPy を基盤として採用している．

Matplotlib：本シミュレータの特徴は，単に数値を計算するだけでなく，食堂レイアウト上で人流をアニメーション表示し，UI から条件を切り替えられる点にある．そのため Matplotlib を中心に実装している．

・pyplot：図と軸の作成，散布図・線分描画などを行う中心モジュールである．

・patches：壁・机・棚など，矩形・多角形などの図形を描くために利用する．

・Line2D：凡例を自作したり，線表現を統一する場合に利用する．

・FuncAnimation：本シミュレータは，各時間ステップで「位置更新 → 描画更新」を繰り返す．この更新処理の枠組みとして FuncAnimation を用いる．

・widgets：CheckButtons（表示切替），Button（再生/停止，リセット，保存などの操作），RadioButtons（日付選択，レイアウト選択（A/B/original）など排他的な切替），RectangleSelector（領域選択）のように，UI 操作を Matplotlib 上で完結させることで，外部 GUI（PyQt 等）を使わずに実験・観察を行える利点がある．

・FFMpegWriter：アニメーションを mp4 等で保存するための機能であり，結果提示や比較動画作成に利用できる．

Pandas：入退場人数データは CSV として与えられるため，読み込み・整形・日付抽出などに Pandas を用いる．シミュレーションの入力条件を UI で切り替えるためにも，表形式データ処理が不可欠である．

datetime：CSV の時刻とシミュレーション内部時刻（仮想時間）を対応づけるために使用する．

random：着席時間のランダム化，ウォーターサーバ立ち寄りの確率分岐など，人の行動の多様性を表現するために利用する．

copy：レイアウト定義や中継点リストなどを複製して安全に編集する際に利用する．

itertools：衝突判定や近接判定で「全ペアの組合せ」を走査する場合に利用する．

layout：食堂の壁・机・椅子・オブジェクト・中継点などの座標情報は，シミュレーションロジックから分離して layout.py にまとめられている．これにより，レイアウト/original/A/B の切替や改修を行う際に，エージェントロジックや描画処理を変更せずに済み，保守性と再利用性が向上する．

（※文責: 菊地皓太）

A.4.1.1　日本語フォント設定

```
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Hiragino Sans', 'Yu Gothic', 'Meiryo',
    'TakaoPGoic', 'IPAGothic', 'Noto Sans CJK JP']
plt.rcParams['axes.unicode_minus'] = False
```

Matplotlib は環境によって日本語が文字化けすることがある．そこで，plt.rcParams を用いて日本語フォント候補を複数指定し，軸ラベル・凡例・UI 文字列を日本語で表示できるようにしている．また，axes.unicode_minus=False を設定し，負号（−）が文字化けして「□」になる問題を防いでいる．この設定を冒頭に置くことで，以降の描画処理・UI 表示において日本語が安定して表示され，デバッグや結果提示を円滑に行うことができる．

（※文責: 菊地皓太）

A.4.2　パラメータ設定

本シミュレータでは，挙動・物理・描画の設定値と，シミュレーション中に更新される「状態」を，コード冒頭の「パラメータ設定」に集約している．これにより，値の調整や挙動の比較を，コード全体を追い回すことなく行えるようにした．

（※文責: 菊地皓太）

A.4.2.1　パラメータ設定の概要

コード冒頭の「パラメータ設定」に集約している変数はすべてが「固定の設定値」ではなく，性質の異なる 2 種類が混在している．

1. シミュレータの性質を決める「調整対象の定数」としての設計パラメータここでいう「定数」とは，実行中に勝手に増減しない値であり，シミュレーションの質（再現性・自然さ・安定性）を左右する設計パラメータである．代表例として，以下が挙げられる．

・エージェントの目標速度，出口へ向かう際の速度倍率
・目標到達判定の閾値
・斥力（障害物や他者）に関する強さ・減衰・有効範囲などの係数
・アニメーション更新間隔や仮想時間倍率
・描画半径や斥力点の配置間隔

これらのように，可視化される調整だけではなく，以下のような実験の前提条件にも関連する．

混雑の再現性：斥力が強い/弱い，距離範囲が広い/狭いだけで，「列が詰まる」「すり抜ける」「壁際で引っかかる」などの現象が変化する．つまり，混雑緩和策の効果を比較するには，これらのパラメータの固定が必須である．

数値安定性：時間刻み（更新間隔）が粗いと，運動方程式の更新が荒くなり，速度が跳ねたり，衝突判定が不自然になったりする．逆に細かすぎると計算負荷が増え，アニメーションが重くなる．

定数は「自然さ」と「計算負荷」のトレードオフの関係にあると言える．
2. シミュレーション中に変化する「状態」としてのパラメーター一方で，変数の中には，実行中に常に更新されるものがある．これは「設定値」というより，シミュレーションが進行するために必要な共有状態（グローバルな状態管理領域）である．代表例として，以下が挙げられる．
・各サービス地点の待ち行列（エージェントの格納リスト）
・待ち行列での並び位置（目標座標）を保持する辞書
・キュー内の通し番号（並び順）を割り当てるカウンタ
・エージェント ID と通し番号の対応表
　これらは，「今この瞬間，どの列に誰が並んでいるか」「誰が何番目か」「次に進めるのは誰か」といった情報を保持するために必須であり，ステップ更新のたびに変化する．状態が一箇所で管理されていないと，同じ列を複数の場所で別々に持ってしまう事故が起こりやすくなる．例えば，「列から抜けたのに辞書に残る」「並び順がずれて重なる」「前詰めしたのに目標座標が更新されない」など，混雑評価にとっての不整合につながる．

（※文責: 菊地皓太）

A.4.2.2　待ち行列関連パラメータ

```
wait_row_1=[]    # セットのエージェントを格納
wait_row_2=[]    # カレーのエージェントを格納
wait_row_3=[]    # ラーメンのエージェントを格納
wait_row_4=[]    # レジの待ち行列
wait_row_5=[]    # レジの待ち行列
wait_row_6=[]    # 食器返却口の待ち行列
wait_row_7=[]    # ウォーターサーバーのエージェントを格納
wait_testrow=[] # テスト用
row_target_1={} # セットの待ち行列
row_target_2={} # カレーの待ち行列
row_target_3={} # ラーメンの待ち行列
row_target_4={} # レジの待ち行列
row_target_5={} # レジの待ち行列
row_target_6={} # 返却口の待ち行列
row_target_7={} # ウォーターサーバの待ち行列

g_queue_counters = { "row_1": 0, "row_2": 0, "row_3": 0, "row_4": 0, "row_5": 0, "row_6": 0, "row_7": 0 }          # 各キューのカウンター
g_queue_agent_map = { "row_1": {}, "row_2": {}, "row_3": {}, "row_4": {}, "row_5": {}, "row_6": {}, "row_7": {} } # エージェント ID とカウンター番号を紐付ける辞書
```

　各 wait __ row は「そのサービス地点で待っているエージェント（Customer オブジェクト）を格納するリスト」である．例えば，食事提供（セット・カレー・ラーメン）や，レジ，食器返却口，

ウォーターサーバーに対応している．これは「先頭から処理される」というキューの性質（FIFO）と相性がよい点や，index() を用いて「自分が何番目か」を取得でき，並び位置の計算に直結する点からリストが使用されている．

各 row __ target はキュー位置（並び座標）が割り当てられている．待ち行列を「ただのリスト」にすると，先頭が抜けた瞬間に index が詰まり，後続全員の index が変化する．そこで本実装では，「並び位置」を別辞書に保持し，各エージェントが参照する「自分の立ち位置」を安定させる設計を導入した．

安定した「番号」でキューを管理する仕組みとして，「キューで新しく並ぶ人に付与する通し番号」を管理する g __ queue __ counters と，エージェント ID と通し番号の対応表である g __ queue __ agent __ map を実装した．この方式のメリットは，リスト順（index）と独立に「個人に紐づく番号」を維持できる点にある．先頭が抜けても残りのエージェントの滞在するべき位置の関係が崩れにくい．

（※文責: 菊地皓太）

A.4.2.3　シミュレーション全体設定パラメータ

```
SELECTED_DATE = ''                  # Count.csv から読み込む日付を選択
VIRTUAL_TIME_SCALE = 3.5            # 仮想時間の速度を調整
SIMULATION_START_TIME = "1130"      # シミュレーションを開始する時刻（CSV の開始時刻に合わせる）
ANIMATION_INTERVAL = 20             # アニメーションの更新間隔（ミリ秒）
```

本シミュレータにおけるシミュレーション全体設定パラメータとは，エージェント個々の性質ではなく，シミュレーション全体の動作条件・時間の扱い・入力データの参照方法・描画更新の進み方を決める設定値のことである．「このシミュレーションを どの条件で」「どの時間スケールで」「どのデータに基づいて」「どの周期で更新して」実行するかを統一的に指定するためのパラメータ群と言える．

SELECTED __ DATE：Count.csv の「どの日付」を再現するか

入場人数の時系列データは CSV から読み込み，日付ごとに分割して保持する設計になっている．ユーザが UI で対象日を選び，その日付キーを SELECTED __ DATE に入れて参照する．

VIRTUAL __ TIME __ SCALE：現実時間と仮想時間の倍率

実行時には，実時間（アニメ－ションのフレーム進行）に対して，仮想時間（シミュレータ内部の時刻）が何倍で進むかを調整する．この倍率は「入場スケジュール（CSV の 1 分刻み）をどれくらいの速度で再生するか」に影響するため，混雑ピークの観察や比較実験の利便性を向上させる．

SIMULATION __ START __ TIME：入場データの基準時刻

SIMULATION __ START __ TIME=”1130”は，CSV の時刻列の基準（11:30 開始）に合わせている．これは収集データ側でも計測時間帯が 11:30 からとなっているため，データ整合のための基準時刻として定義されている．

ANIMATION __ INTERVAL：アニメーション更新周期

ANIMATION __ INTERVAL=20[ms] は Matplotlib に渡される更新間隔であり，見た目の滑らかさだけでなく，内部で運動方程式更新にも使われる．

（※文責: 菊地皓太）

A.4.2.4　エージェント挙動設定パラメータ

```
INITIAL_POSITION = np.array([40.0, 0.0])                    # エージェントの初期位置
GOAL_POS = np.array([50.0, 0.0])                            # エージェントの最終目標地点
AGENT_DESIRED_SPEED = 20                                    # エージェントの目標移動速度
TARGET_REACH_THRESHOLD = 0.5                                # エージェントが目標地点に到達した
    ↪ とみなす距離の閾値
WAYPOINT_RANDOMNESS = 0.5                                   # 中継点の到達判定を緩め、移動をラ
    ↪ ンダムに見せるためのオフセット半径
MIN_CHAIR_WAIT_SECONDS = 30                                 # 椅子の最小待機時間
MAX_CHAIR_WAIT_SECONDS = 40                                 # 椅子の最大待機時間
WATER_SERVER_PROBABILITY = 0.58                             # ウォーターサーバーへ立ち寄る確率
POST_SEATING_WATER_SERVER_PROBABILITY = 0.58                # 着席後にウォーターサーバーへ立ち
    ↪ 寄る確率
CHAIR_SEARCH_START_INDEX = 7                                # 椅子を探し始める中継点のインデッ
    ↪ クス
EXIT_SPEED_MULTIPLIER = 1.2                                 # 出口へ向かう際の目標速度の倍率
EXIT_AREA_X_MIN = 48                                        # 出口エリアの X 座標の最小値
EXIT_AREA_X_MAX = 52                                        # 出口エリアの X 座標の最大値
EXIT_AREA_Y_MAX = 1                                         # 出口エリアの Y 座標の最大値
MAX_SPEED_MULTIPLIER = 1.2                                  # 目標速度に対する最大速度の倍率
COLLISION_IMMUNITY_SECONDS = 0.2                            # 入場後の当たり判定（斥力）が無効
    ↪ 化される時間
```

本シミュレータにおけるエージェント挙動パラメータとは，「エージェントが食堂内で どこへ・どの順番で・どれくらい待って・どの程度の確率で寄り道するか」といった，行動の意思決定を規定する設定値のことである．Social Force Model の斥力や運動方程式が「その瞬間の力学」を決めるのに対し，挙動パラメータは 目的地の選び方・状態遷移・待機時間・確率分岐といった行動上のルールを決定する．具体的には，エージェントがシミュレーション中に辿る一連のプロセスの，その各段階で「次のターゲットをいつ切り替えるか」「到達をどの条件で判定するか」「自然なばらつきをどう入れるか」を制御するのが挙動パラメータである．

INITIAL __ POSITION・GOAL __ POS：入退場座標

本シミュレーションでは，各エージェントが「どこから現れて，どこへ消えるか」を入口・出口の座標として明示的に持たせている．また入口が点 1 つになっているのはモデルを簡潔にするためだが，混雑時には入口付近で重なりが起きやすい．そこで後述の当たり判定無効時間と組み合わせて，湧き出し直後の不安定さを抑える工夫にもつながる．

AGENT __ DESIRED __ SPEED：目標速度

Social Force Model では「目標方向へ向かう駆動力」が基本の推進源になる．典型的には，目標方向の単位ベクトル e を作り，目標速度 v __ des = desired __ speed × e と置き，現在速度 v と

の差を，緩和時間 τ で追従させるという形で，駆動力が (v __ des − v)/τ に比例して計算される．このとき AGENT __ DESIRED __ SPEED は，「そのエージェントが出したい巡航速度」の基準値として，移動全般のテンポを左右する．

EXIT __ SPEED __ MULTIPLIER：退場加速度

本実装では，最終目的地（退場）に向かう段階で EXIT __ SPEED __ MULTIPLIER を掛け，退場時だけ目標速度を意図的に上げている．これは「出口に近づいた人は迷いが減って歩行が速くなる」「退場が遅いと場内滞留が増え，流動が詰まって見える」など，見た目・流量の両面で効く調整である．

TARGET __ REACH __ THRESHOLD：目標到達判定

中継点（waypoint）は点座標で与えられるが，連続空間を数値積分で進む以上，ぴったり同一点に一致する瞬間はほぼ起きない．そこで「この半径内に入ったら到達とみなす」という許容誤差が必要になる．この閾値が TARGET __ REACH __ THRESHOLD であり，コードでは 0.5 が設定されている．

WAYPOINT __ RANDOMNESS：中継点到達判定

中継点を完全に同一点に固定すると，全エージェントが同じ点へ吸い寄せられ，軌跡が束になって「一本の線」に見えやすい．これは現実の歩行（人ごとの微妙な回避・視線・歩幅の差）と比べて不自然で，流動の見た目が硬くなる．そこで通常の中継点では，半径 0.5 以内でランダムなオフセットを加え，同じルートを通っても微妙にズレた軌道になるようにしている．これにより「密集時の局所衝突を減らす」「軌跡に幅が出ることで，混雑の様子が視覚的に理解しやすくなる」といった，流動表現の安定化が可能になる．

一方で，椅子・ウォーターサーバ・キュー目標などは「そこに行けたかどうか」がイベント（着席開始，給水開始，行列位置確定）に直結するため，オフセットを入れると判定がずれてしまう．よって，これらは「特殊ターゲット」として乱数を適用せず，厳密に狙う点として実装している．

MIN __ CHAIR __ WAIT __ SECONDS・MAX __ CHAIR __ WAIT __ SECONDS：着席待機時間

椅子に到達した後は「滞在状態」に入り，その滞在時間を一様乱数で与える．これにより，全員が同じ秒数で立ち上がる不自然さを避けつつ，平均滞在が揃うので解析もしやすくなる．

WATER __ SERVER __ PROBABILITY・POST __ SEATING __ WATER __ SERVER __ PROBABILITY：ウォーターサーバ立ち寄り確率

ウォーターサーバへの立ち寄りは「全員が必ず寄る」行動ではなく，個人差の大きい偶発イベントとして扱う．そこで本実装では確率イベントとして分岐させ，着席前と着席後の 2 つを用意し，同じ給水でも「食事前に水を取りに行く人」と「食事後に水を取りに行く人」を別々に表現できるようにしている．また，この確率は食堂の計測時に観察・集計した結果，ウォーターサーバーに立ち寄った利用者が全体の 58 %であったという実測データを反映して設定している．

EXIT __ AREA __ X __ MIN/X __ MAX・EXIT __ AREA __ Y __ MAX・EXIT __ AREA __ Y __ MAX：退場判定の安定化

最終目標は点（GOAL __ POS）だが，点到達だけで退場扱いにすると「誤差で到達しない」「出口付近で小さく振動して残留する」といった不具合が起きやすい．そこで本実装では，退場判定を「出口付近の矩形領域に入ったら退場」に拡張している．

MAX __ SPEED __ MULTIPLIER：速度上限

Social Force は「駆動力（目標へ行きたい）」と「斥力（障害物や他者を避けたい）」を足し合わせるモデルである．混雑時には斥力が急増し，合力が瞬間的に大きくなって速度が不自然に跳ねる

ことがある（人間の歩行なら“踏ん張る”“止まる”が入るが，単純な運動方程式だと加速してしまう）．そこで更新後の速度に上限を設け，最大速度を声ね異様に制御している．この上限は「速さを抑える」だけでなく，瞬間ワープのような移動や壁抜けに見える挙動を減らし，視覚的な破綻を防ぐためにも機能する．

COLLISION __ IMMUNITY __ SECONDS：入場直後の斥力無効化

エージェントが入場（アクティブ化）してから一定時間だけ「他エージェントとの当たり判定（Social Force 由来の斥力）」を無効化するためのパラメータである．本シミュレータでは，入口付近にエージェントが短時間に集中して生成されるため，入場直後から他者との斥力を有効にすると，入口周辺での不自然な跳ねや振動が起こる．生成直後はエージェント間距離が極端に小さくなりやすく，Social Force が急激に大きくなって挙動が荒れたり，群れが入口で弾け飛ぶような動きになりやすい．

（※文責: 菊地皓太）

A.4.2.5　物理モデルパラメータ

```
AGENT_MASS = 0.01                    # エージェントの質量（慣性に関連）、小さいほど機敏に動く
RELAXATION_TIME = 0.05               # 速度緩和時間、目標速度に達するまでの時間
REPULSION_FORCE_STRENGTH = 5         # 障害物からの斥力の基本強度 (A)
REPULSION_FORCE_DECAY = 0.3          # 斥力が影響する距離の大きさ (B)
REPULSION_EFFECTIVE_RANGE = 2.0      # 斥力が働き始める最大距離
SOCIAL_FORCE_STRENGTH = 17           # 他エージェントからの斥力の基本強度
SOCIAL_FORCE_DECAY = 0.5             # 斥力が影響する距離の大きさ
SOCIAL_FORCE_EFFECTIVE_RANGE = 2.0   # 斥力が働き始める最大距離
```

本シミュレータの移動は Social Force Model の考え方に基づき，「目的地へ向かう希望推進力」と「障害物・他者から距離を取る反発力」を合成して毎フレーム更新する．これらの力学的な動きの質を決める定数群が，物理モデルパラメータである．

AGENT __ MASS：質量

力が速度変化へ変換されるスケールを決める．一般に質量が大きいほど慣性が強くなり，外力（斥力や駆動力）に対して速度が変わりにくい．逆に小さいほど機敏に反応する．本実装では 0.01 と非常に小さめに設定し，狭い通路・混雑環境でも「詰まりを回避しようと細かく動く」挙動が出やすいようにしている．

RELAXATION __ TIME：速度緩和時間

目標速度に達するまでの時間スケールを表す．値が小さいほど目標速度へ素早く追従し，動きは円滑になるが，混雑時は力が過剰に効いて速度が揺れやすい．値が大きいほど追従が遅くなり，滑らかになる一方，混雑で流れが変わっても反応が鈍くなる．

REPULSION __ FORCE __ STRENGTH：斥力強度

障害物からの押し返しの最大級の強さを決める．大きいほど壁際への接近を強く拒み，通路中央へ寄る傾向が強まる．一方，強すぎると狭い場所で押し合いが発生し，振動や足踏みが増える原因

にもなる．

REPULSION __ FORCE __ DECAY：斥力減衰距離

距離が離れたときに斥力がどれくらい早く弱まるかを決める．小さいほど「近づいた瞬間だけ強烈に効く」尖った壁になり，大きいほど「遠くからじわじわ効く」柔らかい壁になる．

REPULSION __ EFFECTIVE __ RANGE：斥力有効距離

斥力点との距離がこの値より大きい場合は計算対象から除外し，計算量と不必要な遠距離干渉を抑える．

SOCIAL __ FORCE __ STRENGTH：エージェントからの斥力強度

対人斥力の基本強度であり，障害物よりも強めに設定されている．これは，人は壁よりも他者との接触を強く避けるという直感を，モデル側で強度差として表現する狙いがある．

SOCIAL __ FORCE __ DECAY：エージェントからの斥力減衰距離

対人斥力が距離とともに弱まる速さを決める．値が小さいと接触寸前で急激に回避し，値が大きいと少し離れた段階から回避が始まる．

SOCIAL __ FORCE __ EFFECTIVE __ RANGE：エージェントからの斥力有効距離

遠距離の相互作用を切り捨て，局所相互作用として扱うための上限距離である．

（※文責: 菊地皓太）

A.4.2.6　描画設定パラメータ

```
REPULSION_POINT_RADIUS = 0.25       # 斥力点の描画半径
REPULSION_POINT_STEP_FACTOR = 1     # 斥力点の配置間隔
CHAIR_RADIUS = 0.3                  # 椅子の描画半径
AGENT_RADIUS = 0.4                  # エージェントの描画半径
```

本シミュレータにおける描画設定パラメータは，シミュレーション結果の可視化（アニメーション）を分かりやすくするだけでなく，障害物回避の当たり判定の細かさ（斥力点の配置密度）にも影響する．

REPULSION __ POINT __ RADIUS：斥力点の描画半径

斥力点を円として描画する際の半径である．見た目の太さだけでなく，結果的に「壁の点群がどれだけ細かく敷き詰められるか」を左右する．

REPULSION __ POINT __ STEP __ FACTOR：斥力点の配置感覚

半径に対してどれくらいの間隔で点を置くかを決める係数である．1 の場合，概ね半径と同程度の刻みで点が配置される．

斥力点は「描画のためだけ」ではなく，wall __ positions として座標列が保持され，エージェントの回避計算そのものに利用される．したがって，点間隔が粗すぎると壁に隙間が生じ，局所的に貫通しやすくなる．一方で細かすぎると，斥力計算対象が増え計算負荷が上がるため，貫通しない最小限の密度を狙って調整する方針とした．実際にメイン処理でも，斥力点を計算・描画して wall __ positions を得る流れになっている．

CHAIR __ RADIUS：椅子の描画半径

椅子を円として描画する際の半径である．レイアウト上で「座席の占有感」を視覚的に表現し，座席周辺の混雑や座席探索行動が理解しやすくなる．

AGENT__RADIUS：エージェントの描画半径

エージェントを円として描画する半径である．本実装では，衝突回数の計測において「エージェント間距離が AGENT__RADIUS 未満なら衝突」と判定しており，見た目の大きさがそのまま評価指標の閾値としても機能する．

AGENT__RADIUS は単なる見た目調整ではなく，「混雑度合いをどう定義するか」にも直結する．したがって，衝突回数を比較する実験では，AGENT__RADIUS をむやみに変更せず，変更する場合は衝突の定義自体が変わることを明記した上で扱う必要がある．

（※文責: 菊地皓太）

A.4.3 描画関数

食堂レイアウト（壁・障害物・固定オブジェクト・テーブル・椅子）は描画関数群を用いて実装し，Matplotlib 上に描画を行った．描画は Axes オブジェクト ax に対して行い，ax.scatter およびパッチ（patches.Rectangle, patches.Circle）を用いた構成要素から成る．

（※文責: 菊地皓太）

A.4.3.1 描画関数の概要

描画関数は大きく分けて以下の 3 つの系統から構成される．

・固定オブジェクトの描画

レジ・入口・出口・返却口・ウォーターサーバー・ゴミ箱・テラス出入口など，位置が「点」として扱えるオブジェクトは ax.scatter() により描画する．これらは視認性向上のため星形マーカー（marker=’*’）とし，オブジェクト種別ごとに色を変えて区別する．

・テーブルと座席の描画

テーブルは矩形（patches.Rectangle），椅子は円（patches.Circle）として描画する．これにより，占有領域（テーブル）と座席位置（椅子）を直感的に表現できる．

・壁や障害物と斥力点の生成と描画

Social Force Model 等の衝突回避を実現するため，壁や障害物の境界上に「斥力点」を離散配置する．壁（線分）と障害物（矩形）の両方を扱えるようにし，生成した点群は後段の力計算にも利用できるよう返り値として返す．

（※文責: 菊地皓太）

A.4.3.2 線分と矩形の壁の描画

```
def draw_walls_and_rects(ax, obstacles_data, radius, step_factor):
        if obs.get("type") == "line":
            start, end = obs["start"], obs["end"]
            length = np.linalg.norm(end - start)
            steps = max(int(length / step), 1)
        elif obs.get("type") == "rect":
            x0, y0 = obs["pos"]
```

```
            w, h = obs["width"], obs["height"]
            for x in np.arange(x0, x0 + w + step, step):
                for y in [y0, y0 + h]:
                    pos = np.array([x, y])
                    wall_circles_pos.append(pos)
            for y in np.arange(y0 + step, y0 + h, step):
                for x in [x0, x0 + w]:
                    pos = np.array([x, y])
                    wall_circles_pos.append(pos)
    for pos in wall_circles_pos:
        circ = patches.Circle(pos, radius=radius, facecolor="black",
        ↪  edgecolor="black", zorder=2)
        ax.add_patch(circ)
        rect_circles_patches.append(circ)
```

本シミュレーションにおける障害物としての壁は，連続した線や面として直接描画するのではなく，境界上に等間隔で配置した円形オブジェクトの集合として表現している．これにより，壁や障害物の形状を柔軟に扱うことが可能となり描画処理を統一的に実装できる．

壁・障害物の描画は draw __ walls __ and __ rects(ax, obstacles __ data, radius, step __ factor) により行われる．入力として与えられる obstacles __ data は，線分で表される壁および矩形で表される壁の定義情報を要素とする辞書の配列であり，以下の 2 種類で実装される．

・線分状の壁の描画（type = ”line”）

線分として定義された壁は，始点（start）と終点（end）により表現される．まず，始点から終点までの距離を計算し，その長さに応じて分割数 steps を決定する．これにより，壁の長さに比例した密度で描画点が配置されるようになっている．

・矩形状の障害物の描画（type = ”rect”）

矩形として定義された障害物は，左下座標 pos と幅 width，高さ height により表現される．矩形障害物については，内部を塗りつぶすのではなく，四辺の境界のみを描画対象としている．この実装により，矩形の四辺が等間隔の点列として表現され，障害物の輪郭を明確に可視化できる．

生成された壁および障害物の境界点は，円形オブジェクトとして描画される．各境界点は patches.Circle を用いて描画され，壁や障害物は複数の円が連なった形状として図中に表示される．これにより，線分状の壁および矩形状の障害物を，統一的かつ視覚的に分かりやすい方法で描画している．

（※文責: 菊地皓太）

A.4.3.3　固定オブジェクトの描画

```
def draw_registers(ax, register_positions_data):
    """レジを描画する"""
```

```
    return ax.scatter(register_positions_data[:, 0],
    ↪  register_positions_data[:, 1], color='blue', s=130, marker='*',
    ↪  zorder=4)


def draw_entrance(ax, entrance_pos):
    """入口を描画する"""
    return ax.scatter(entrance_pos[0], entrance_pos[1], color='orange',
    ↪  s=130, marker='*', zorder=4)


def draw_exit(ax, exit_pos):
    """出口を描画する"""
    return ax.scatter(exit_pos[0], exit_pos[1], color='green', s=130,
    ↪  marker='*', zorder=4)


def draw_return_box(ax, return_box_pos):
    """食器返却口を描画する"""
    return ax.scatter(return_box_pos[0], return_box_pos[1], color='purple',
    ↪  s=130, marker='*', zorder=4)


def draw_water_server(ax, water_server_pos):
    """ウォーターサーバーを描画する"""
    return ax.scatter(water_server_pos[0], water_server_pos[1],
    ↪  color='deepskyblue', s=130, marker='*', zorder=4)


def draw_trash_box(ax, trash_box_pos):
    """ゴミ箱を描画する"""
    return ax.scatter(trash_box_pos[0], trash_box_pos[1], color='brown',
    ↪  s=130, marker='*', zorder=4)


def draw_terrace_exit(ax, terrace_exit_pos):
    """テラスの出入口を描画する"""
    return ax.scatter(terrace_exit_pos[0], terrace_exit_pos[1],
    ↪  color='orangered', s=130, marker='*', zorder=4)
```

本シミュレーションでは，レジ・入口・出口・返却口・ウォーターサーバー・ゴミ箱・テラス出入口といった固定オブジェクトを，平面上の特定座標に存在する「目印（ランドマーク）」として扱

う．これらのオブジェクトは，壁やテーブルのように面積（占有領域）を厳密に描くことよりも，「どこにあるか」を明確に示すことが重要であるため 1 点（または複数点）の座標として表現し，散布図（scatter）で描画している．

固定オブジェクトの描画はすべて ax.scatter() により行う．scatter は「点を描く」ための関数であり，座標を与えるだけで簡潔に描画できるためオブジェクトのような位置を示す要素の可視化に適している．また，scatter の返り値（PathCollection）は描画オブジェクトそのものであり，凡例作成や表示制御にも利用可能である．レジのみ複数箇所存在することを想定しており，register __ positions __ data は (N,2) の NumPy 配列で与える．このように列参照することで，複数レジを一度の scatter 呼び出しで描画でき，描画コストやコード量を抑えられる．

入口：draw __ entrance(ax, entrance __ pos)

出口：draw __ exit(ax, exit __ pos)

返却口：draw __ return __ box(ax, return __ box __ pos)

ウォーターサーバー：draw __ water __ server(ax, water __ server __ pos)

ゴミ箱：draw __ trash __ box(ax, trash __ box __ pos)

テラス出入口：draw __ terrace __ exit(ax, terrace __ exit __ pos)

これらは 1 箇所のみを想定し，入力は [x, y] または (x, y) の 2 要素座標で与える．scatter には pos[0]（x）と pos[1]（y）を渡すことで，単一点として描画している．

サイズ：s=130

scatter の s はマーカー面積を指定する．テーブルや椅子がパッチとして描画される一方，オブジェクトは点描画であるため，点が小さいと見落とされやすい．そこで 大きめの面積（130）を設定し，オブジェクトが直感的に目立つようにしている．

形状：marker=’*’（星形）

オブジェクトを星形にすることで，四角（テーブル）や円（座席）と視覚的に混同しにくくなる．つまり，形状差によって「オブジェクト」「テーブル」「座席」「壁」を瞬時に判別できるようにしている．

描画順：zorder=4

zorder は重なり順を決める値であり，数値が大きいほど前面に描画される．固定オブジェクトはレイアウト上の重要情報であるため，常に最前面（zorder=4）に描画し，壁・テーブル・椅子等に隠れないようにしている．

色分け

本実装ではオブジェクトごとに color を固定しており，色を見ただけでオブジェクト種類を判別できる．

レジ：blue

入口：orange

出口：green

返却口：purple

ウォーターサーバー：deepskyblue

ゴミ箱：brown

テラス出入口：orangered

このような色分けは，レイアウト図の説明文や凡例（legend）と組み合わせることで，初見の閲覧者でも理解しやすい可視化となる．

各関数は ax.scatter() の返り値をそのまま返す．返り値は Matplotlib の描画オブジェクトであ

るため，例えば以下の用途に利用できる．
・凡例（legend）用のハンドルとして利用
・表示か非表示かの切り替え（可視性制御）
・アニメーション時に再描画せず参照し続ける（更新対象の管理）
　つまり，単なる描画処理だけではなく，可視化の管理を行いやすい形で返している点が設計上の特徴である．

（※文責: 菊地皓太）

A.4.3.4　テーブルの描画

```
def draw_tables(ax, tables_data):
    """テーブルを描画する"""
    drawn_objects = []
    for table in tables_data:
        rect = patches.Rectangle(table["pos"], table["width"],
        ↪  table["height"],
                                 edgecolor='black', facecolor='lightblue',
                                 ↪  zorder=2)
        ax.add_patch(rect)
        drawn_objects.append(rect)
    return drawn_objects
```

　固定オブジェクトが「位置を示す点」であるのに対し，テーブルは人が滞在する場所に関わるため，占有領域を図形として明示することが重要である．そこで本実装では，テーブルを矩形としてパッチ（Patch）で描画する．
テーブルの描画
　テーブルは面積を持つオブジェクトであり，単なる点ではなく「どの範囲がテーブルで占有されているか」を示す必要がある．そのため，patches.Rectangle を用いて矩形領域（占有領域）として描画している．tables __ data はテーブルを複数扱えるようにした辞書のリストであり，各要素が 1 つのテーブル定義に対応する．各辞書は次のキーを持つ．・pos：矩形の基準座標（通常は左下の (x, y)）
・width：横幅
・height：縦幅
　このように「座標とサイズ」で定義することで，レイアウト変更時にパラメータの差し替えだけで配置を変更でき，実装の再利用性が高い設計とした．
描画処理の流れ
・drawn __ objects を用意して，後で描画オブジェクトをまとめて管理できるようにする．
・tables __ data を 1 つずつ取り出し，各テーブルに対して矩形パッチ Rectangle を生成する．
・ax.add __ patch(rect) で Axes に追加し，実際に図として表示する．
・生成したパッチを drawn __ objects に格納して返す．
描画パラメータ

・edgecolor=’black’

テーブルの境界線を黒で描くことで，背景や他の家具と重なった場合でもテーブルの輪郭が失われにくくなる．特に facecolor を淡色にしている場合，境界線がないと輪郭が曖昧になりやすいため，輪郭強調は視認性の観点で重要である．

・facecolor=’lightblue’

テーブル面を薄い青で塗ることで，「この領域はテーブルが占有している」ことが一目で分かる．濃色ではなく淡色にしているのは，テーブルが主役ではなくレイアウト要素であり，上に重なる椅子やエージェントを邪魔しないためである．

・zorder=2

描画順序を表す zorder を 2 に設定し，後から描画する椅子や固定オブジェクトがテーブルに埋もれないようにしている．つまり，テーブルは「土台」として背景寄りの層に置かれる設計である．

返り値（drawn __ objects）

この関数は，描画した Rectangle パッチのリストを返す．これにより，後処理として次が可能になる．

・表示のオン/オフ（可視性切り替え）

・色や透明度の変更（混雑度に応じた色変更等）

・アニメーション更新時に参照し続ける（再生成を避ける）

描画するだけではなく，「描画したオブジェクトを後で制御できる」設計になっている点が本実装の特徴である．

（※文責: 菊地皓太）

A.4.3.5　椅子の描画

```
def draw_chairs(ax, chair_positions_data, radius, color='black'):
    """椅子を描画する"""
    drawn_objects = []
    for pos in chair_positions_data:
        circle = patches.Circle(pos, radius=radius, facecolor=color,
                                edgecolor='black', zorder=3)
        ax.add_patch(circle)
        drawn_objects.append(circle)
    return drawn_objects
```

固定オブジェクトが「位置を示す点」であるのに対し，椅子は人が滞在する場所に関わるため，座席位置を図形として明示することが重要である．そこで本実装では，椅子を円としてパッチ（Patch）で描画する．

椅子の描画

椅子は座席の位置を示す重要な情報であるが，点で描くと小さすぎて視認しづらく，またテーブル等との関係も掴みにくい．そこで本実装では patches.Circle を用いて，椅子を円形の図形として描画している．chair __ positions __ data は椅子の座標を並べた配列（リストや NumPy 配列）であり，各辞書は次のキーを持つ．

・(x, y) の 2 要素座標

椅子は数が多くなりやすいため，テーブルと同様に「データ側で座標列を定義し，描画関数はそれを描くだけ」という役割分担になっている．

描画処理の流れ

・椅子座標を 1 つずつ取り出す．

・円パッチ Circle を生成する．

・ax.add __ patch で追加して描画する．

・作成したパッチをリストに入れて返す．

描画パラメータ

・radius

円の半径を引数で受け取ることで，描画スケールに応じて椅子の見た目を調整できる．座標系の単位が「メートル相当」なのか「セル単位」なのかで適切な椅子サイズは変わるため，固定値ではなく外部から指定できるようにしている．

・facecolor=color（デフォルトは black）

デフォルトを黒にしているのは，椅子を輪郭だけでなく塗りつぶしで表示し，座席が背景に埋もれないようにするためである．また，引数 color を受け取る設計により，「利用中の椅子だけ色を変える」など拡張が容易になる．

・zorder=3

椅子はテーブルの上にある（前面に見える）べき情報であるため，zorder=3 としてテーブル（zorder=2）より前面に描画する．これにより，椅子がテーブルの色に埋もれず，座席配置が明確に把握できる．

テーブルと椅子を分離した設計の利点

・視覚的役割が異なる．

テーブルは占有領域であるのに対し，椅子は座席位置を表す．

・データ構造が異なる

テーブルは位置とサイズ（pos, width, height）であるのに対し，椅子は座標列である．

・拡張が容易

着席中の椅子だけ色変更，テーブル種別で色変更，座席数カウントなどを反映できる．

（※文責: 菊地皓太）

A.4.3.6　斥力点の生成

```
circ = patches.Circle(
  pos,
  radius=radius,
  facecolor="black",
  edgecolor="black",
  zorder=2)
ax.add_patch(circ)
```

Matplotlib では Circle を作っただけでは画面に出力されず，ax.add __ patch() により Axes に

追加して初めて描画される．したがって，「描画」として成立しているのは ax.add __ patch(circ) を呼んだ時点である．

patches.Circle の各引数

・pos（中心座標）

pos は円の中心座標であり，斥力点を配置する位置そのものを表す．壁や障害物の境界上に生成した座標列が pos として渡されることで，「境界に沿って点が並ぶ」見た目になる．

・radius=radius（見た目の大きさ）

radius は円の半径であり，図としての視認性を決める．小さすぎると点が見えず，逆に大きすぎると隣接する点同士が重なって壁が太く見えたり，他のオブジェクトと干渉して見づらくなる．そのため本実装では，配置間隔（step）を半径と連動させる設計になっている．

・facecolor=”black”（塗りつぶす色）

斥力点は「壁・障害物」を表す要素であり，視覚的に最も無機物・境界らしい色として黒を採用している．塗りつぶしを有効にすることで，背景色やテーブル色に対して埋もれにくく，壁の輪郭として認識しやすい．

・edgecolor=”black”（輪郭線）

輪郭線を同じ黒にすることで，塗りつぶしだけの円よりも形状が明確になる．特に円同士が隣接・重なりぎみになる状況では，輪郭がある方が「点列」としての連続性が分かりやすい．

・zorder=2（重なり順）

斥力点は壁・障害物の境界を示すが，入口・出口・レジなどの固定オブジェクト（zorder=4）より前面に出る必要はない．また椅子（zorder=3）よりも前面に出すと座席が見えづらくなるため．背景寄りの zorder=2 としている．

配置間隔（step）と密度設計

斥力点の配置密度は step = radius * step __ factor という式で制御する．この設計の利点は，点の大きさ（radius）を変えたときに，点間隔も一緒に変わる点にある．このように，半径と間隔を比例させることで，壁の見え方（密度感）がスケール変更に対して比較的一定に保たれる．これは「視認性」と「安定性」を両立するための設計である．

返り値としてパッチを保持

斥力点は描画点数が多くなりがちであり，後から以下の操作への拡張に便利である．

・表示/非表示の切り替え

・再描画や削除

・色変更

そのため，生成した Circle を line __ circles __ patches と rect __ circles __ patches に分けて保持している．描画オブジェクトを保持しておくことで，必要に応じて remove() などの操作が可能になり，可視化を制御しやすくなる．

（※文責: 菊地皓太）

A.4.3.7　返り値の利用

本プログラムにおける各描画関数は，単に図を描画するだけでなく，描画したオブジェクトへの参照を返す設計としている．これにより，描画後のオブジェクトを外部から操作可能となり，可視化の柔軟性およびアニメーション処理との親和性が大きく向上する．Matplotlib では，描画関数（scatter, add __ patch など）が返すオブジェクトを保持しておかない限り，後からその描画要素

を個別に操作することは困難である．そこで本実装では，以下の目的のために描画関数が必ず返り値を持つようにしている．
・凡例（legend）への登録
・表示と非表示の切り替え
・アニメーション更新時の再利用や削除
・状態に応じた色や透明度の変更

この設計により，「一度描いて終わり」の静的な描画ではなく，動的に制御可能な描画要素として扱うことができる．設計上では壁・障害物の返り値を，座標（計算用），線分由来パッチ（描画用），矩形由来パッチ（描画用）に分離していることで，
・計算処理と描画処理を論理的に分離
・障害物種別ごとの表示制御
・後処理や拡張の容易さ
を同時に実現している．

（※文責: 菊地皓太）

A.4.3.8　表示順序と視認性

本実装では，レイアウト要素（壁・障害物・テーブル・椅子・固定オブジェクト）を同一平面上に重ねて描画するため，重なり順序（表示レイヤ）を意図的に設計している．Matplotlib ではこの重なり順を zorder により制御することが可能であり，数値が大きいほど前面に描画される．zorder の設計が曖昧なまま描画すると，オブジェクトや壁が設備マーカーを覆い隠したり，椅子がテーブルに埋もれて座席配置が分かりにくくなるなど，可視化の目的を損なう可能性がある．そのため本実装では，各要素の役割に応じて明確に階層を分け，常に読み取りやすい図になるよう調整している．
zorder 設計の基本方針
可視化における要素は大きく次の 3 種類に分けられる．
・背景・占有領域としての要素（床面に固定されたもの，領域を示すもの）
テーブル，壁・障害物（境界表現）
・配置や状態を読み取りたい要素（座席の位置など）
座席
・シミュレーション上の重要地点（目標地点やランドマーク）
入口，出口，レジ，返却口など固定設備
この分類に従い，「背景，中層，最前面」の順に zorder を割り当てている．
テーブル：zorder=2
テーブルはレイアウトの中で面積を持つ要素であり，図の中では「床に置かれた占有領域」として扱う．重要なのは，テーブルそのものを目立たせることではなく，座席やエージェントがどこに存在しているかを読み取れる状態を作ることである．そのためテーブルは zorder=2 として背景寄りに描画し，椅子や固定設備が必ず上に重なって見えるようにしている．また，テーブルは facecolor=’lightblue’ の淡色で塗りつぶしており，背景要素として視認できつつも前面要素の邪魔になりにくい．
斥力点（壁・障害物）：zorder=2

壁や障害物はレイアウトの境界を表す重要な要素である一方，可視化における主役は「人の移動」や「目的地（設備）」である．壁・障害物が前面に出すぎると，設備マーカーや椅子を覆い，視認性が損なわれる．そのため，壁・障害物を表現する円群はテーブルと同じ zorder=2 に置き，背景層として扱っている．これにより，壁は見えるが主情報（座席や設備）を邪魔しない，というバランスを実現している．線分由来の点を逐次描画する場合でも，zorder=2 を固定しているため，壁の表示が前面要素を上書きすることはない．

椅子：zorder=3

椅子は座席位置を示す要素であり，利用者にとって「座席がどこにあるか」「座席が密集しているか」を把握するうえで重要な情報である．ところが椅子がテーブルと同じレイヤ（同じ zorder）にあると，テーブルの塗りつぶしや輪郭に埋もれ，座席が判別しづらくなる．そこで椅子はテーブルより 1 段前の zorder=3 とし，座席が常にテーブルの上に見えるようにしている．この設計により，テーブル配置と椅子配置の関係（どのテーブルにどれだけ座席があるか）が視覚的に把握しやすくなる．

固定オブジェクト：zorder=4

入口・出口・レジ・返却口などの固定オブジェクトは，シミュレーション上の「目標地点」「重要地点」に該当する．これらが見えない状態になると，エージェントがどこへ向かうのか，流れがどの設備に関係しているのかが把握できなくなり，可視化の価値が大きく低下する．そのため固定設備は最前面の zorder=4 に統一し，壁・テーブル・椅子より必ず前に表示されるようにしている．特に星形は，椅子の円やテーブルの矩形と形状が明確に異なるため，形状差によっても識別しやすい．

（※文責: 菊地皓太）

A.4.4　エージェント

本シミュレーションにおいて，歩行者（エージェント）は Customer クラスで実装されている．このクラスはシミュレーション空間内を移動する個々の歩行者の物理的状態（速度，位置）と行動状態（食事中，移動中など）を管理する役割を担っている．

（※文責: 島津諒雅）

A.4.4.1　エージェントパラメータ

```
self.id = agent_id                                  # エージェントのユニーク ID
self.color = color                                  # エージェントの色
self.pos = np.array(start_pos, dtype=float)         # 現在位置
self.velocity = np.zeros(2, dtype=float)            # 現在の速度 (x, y)
selected_waypoints = random.choice(all_waypoints)   # エージェント毎にランダムなルートを選択
self.targets = selected_waypoints + [{"pos": goal, "wait": [0, 0]}]      # 辿るべき中継点のリスト
self.walls = walls                  # 斥力計算に用いる壁の位置
self.params = params                # パラメータを一括で受け取る
```

```
self.entry_frame = entry_frame     # エージェントが登場するフレーム
self.is_active = False             # エージェントがシミュレーションに登場したか
self.is_seated = False             # エージェントが着席中かどうかのフラグ
self.target_index = 0              # 次に目指すターゲットのインデックス
self.active_target_pos = None      # 現在目指している目標座標
self.reached_goal = False          # 最終目標に到達したかどうかのフラグ
self.is_waiting = False            # 中継点で待機中かどうかのフラグ
self.wait_timer = 0                # 待機時間をカウントするタイマー
self.chair_target_added = False    # 椅子へのルートが経路に追加されたかのフラグ
self.target_chair_pos = None       # 目標としている椅子の座標
self.has_left_chair = False        # 椅子から離れたかどうかのフラグ
self.return_route_added = False    # 椅子からの復路が追加されたかのフラグ
self.time_since_active = 0.0       # アクティブになってからの経過時間
self.post_seating_water_server_added = False      # 着席後のウォーターサーバー分岐処理を実行するためのフラグ
```

本シミュレーションにおいて,「エージェント」は食堂を利用する個々の人を抽象化したオブジェクトである．各エージェントは自律的に判断し行動するために，クラスの属性として多数のパラメータを保持している．これらのパラメータは物理演算と移動の制御，行動状態とライフサイクルの管理，個体差と経路の多様性の実現などの役割を担っている．

self.id：エージェントを一意に識別するための ID

他のエージェントとの衝突判定を行う際に，自分以外のエージェントを参照するために用いる．

self.color：描画時にエージェントを表す円の色

各エージェントにランダムに描画する色を与えている．

self.pos：シミュレーション空間上におけるエージェントの現在の位置座標

numpy 配列で保持されており，毎ステップのオイラー法に用いて更新される．

self.velocity：エージェントの現在の速度ベクトル

初期値は 0 に設定されており，運動方程式の F=ma に基づいて計算を行う．この速度ベクトルパラメータを用いて次のエージェントの位置座標を予測する．

selected __ waypoints：エージェントの主なルート

シミュレーションは，主に 3 つの「麺ルート」,「丼ルート」,「セットルート」の 3 つのルートで構成されており，これら 3 つのうちどのルートを選択したかが設定される．それぞれのルートには

self.targets：辿る目標地点のリスト

食堂の入口から出口までのエージェントの移動経路における目標地点をリストである．selected __ waypoints で選択されたルートに加え，エージェントのスタート位置（食堂の入口），エージェントのゴール位置（食堂の出口）を加えたものになっている．

self.walls：壁や障害物の座標データ

エージェントの障害物に対する斥力計算の対象を参照するためのパラメータ．

self.params：質量，目標速度などの物理定数を格納した辞書

7.4.2.4，7.4.2.5 で解説したパラメータを辞書型で格納したもの．これらのパラメータは力の計算で用いられる．

self.entry __ frame：エージェントがシミュレーション上に出現する予定の時刻

実際に計測したデータを参照し，シミュレーション上に描画する予定であるエージェント，それぞれにシミュレーションに描画し始める時刻（フレーム）を設定している．

self.is __ active：シミュレーション上に存在しているかを判定するフラグ

初期状態は False に設定されており，現在のフレーム数が entry __ frame に達した際に True に切り替わる．エージェントの計算・描画をする対象であるかの判定を行う．

self.is __ seated：着席しているかどうかを示すフラグ

このフラグが True である（着席している）時，そのエージェントは他のエージェントの斥力が働く対象とならない．また，そのエージェントの計算は行われずスキップされる．

self.target __ index：目標地点のリストのうち，現在の目標地点はどれか参照するための番号

エージェントの目標地点が格納されているリスト（self.targets）において，現在のエージェントの目標地点の情報を参照するための番号がである．

self.active __ target __ pos：現在目標地点と設定している地点の位置座標

self.target __ index によって参照された目標地点の位置座標が格納されている．Social force model における希望推進力を求める際の，引力の中心としての役割を担っている．

self.reached __ goal：ゴール（食堂の出口）に達したかどうかの判定をするフラグ

ゴールに達したと判定されフラグが True になると，シミュレーション上から削除され，力の計算，描画が行われなくなる．

self.is __ waiting：エージェントが待ち状態であるかどうかの判定をするフラグ

複数の目標地点（料理提供棚，レジ，席など）において，エージェントは一定時間待機をする．待機をしている間は物理演算は行われず位置が固定される．

self.wait __ timer：その地点における待機時間

目標地点それぞれで待機時間が設定されており，その地点に達した瞬間から待機時間を減らしていく．待機時間が 0 になったら待ち状態が解除（self.is __ waiting が False）される．

self.chair __ target __ added：目標地点のリストに座席の情報が格納されたか判定するフラグ

self.targets にエージェントが座ろうとしている座席が目標地点として格納されているかを判定するためのフラグ．これにより，同じエージェントが複数の座席を追加することを防いでいる．

self.target __ chair __ pos：エージェントが着席する座席の位置座標

self.target にはエージェントが着席予定である座席も含まれている．その座席の位置座標が設定されている．

self.has __ left __ chair：食事を終え離席をしたときの処理を管理するためのフラグ

食事を終えエージェントが移動可能状態への切り替えや，エージェントが座っていた座席が空席になったりなど，エージェントが離席したことにより実行する処理を一度に行うためのフラグ，同じ処理を複数回実行することを防ぐ効果もある．

self.return __ route __ added：座席から返却口までのルートが追加されたか判定するためのフラグ

エージェントが食事を終えた瞬間に，self.targets に座席から返却口までの経路を追加する．この処理がきちんと行われたのかを判定する役割を担っている．

self.time __ since __ active：エージェントの食堂の滞在時間

エージェントがシミュレーション上に追加された瞬間から削除されるまでの時間をカウントしている．もし，エージェントが壁にめり込んでしまい動けなくなってしまった場合に，ある程度の時間が経過した際にそのエージェントを削除する処理を行う．これにより，スタックによりシミュレーションが重くなり動作しなくなることを防ぐ効果がある．

self.post __ seating __ water __ server __ added：食事終了後にウォーターサーバへ行くか判定をするフラグ

シミュレーションにおいてウォーターサーバに行くタイミングは食事前と食事後の 2 パターン存在する．このパラメータはそのうちの食事後にウォーターサーバに向かうかを判定するフラグである．

（※文責: 島津諒雅）

A.4.4.2 待ち行列

本シミュレーションにおいて，待ち行列のロジックは「__ set __ next __ target 関数」で記述されている．待ち行列は主に「エージェントを待ち行列に割り当てる」「エージェントを待ち行列から削除」の 2 つのロジックで構成されている．

それぞれの待ち行列は wait __ row __ X（X=1,2,3,...,7）の合計 7 つの待ち行列が定義されている．それぞれの待ち行列は以下の待ち行列を表している．

- wait __ row __ 1…セットルートにおける料理提供待ちの行列
- wait __ row __ 2…カレールートにおける料理提供待ちの行列
- wait __ row __ 3…ラーメンルートにおける料理提供待ちの行列
- wait __ row __ 4…入口側のレジにおける待ち行列
- wait __ row __ 5…ウォーターサーバ側のレジにおける待ち行列
- wait __ row __ 6…食器返却口における待ち行列
- wait __ row __ 7…ウォーターサーバにおける待ち行列

これらの待ち行列は以下の 3 つのパターンに分けられる．

- 1 列または 2 列かを動的に切り替える事ができる待ち行列
  wait __ row __ 1，wait __ row __ 2，wait __ row __ 3
- 1 列固定の待ち行列
  wait __ row __ 4，wait __ row __ 5，wait __ row __ 6
- 一定人数を超過した際にずれて並ぶ待ち行列
  wait __ row __ 7

エージェントを待ち行列に割り当てる処理は以下のコードによって記述されている．

1 列または 2 列かを動的に切り替えることができる待ち行列

```
if np.array_equal(initial_current_target_info["pos"], np.array([3.5, 1])):
            if self not in wait_row_1:
                wait_row_1.append(self)
                # グローバルカウンターを進め，エージェント ID と紐付ける
                g_queue_counters["row_1"] += 1
                g_queue_agent_map["row_1"][self.id] =
                ↪  g_queue_counters["row_1"]
            # 自分の「番号」
```

```
            persistent_idx = g_queue_agent_map["row_1"][self.id]
            # 自分の「現在のリスト上の位置」
            my_list_index = wait_row_1.index(self)
            # 自分の「現在のリスト上の順番」(1始まり)
            idx_for_pos_calc = my_list_index + 1
            if my_list_index != 0:
                if SELECTED_QUEUE_TYPE == '二列':
                    if idx_for_pos_calc % 2 != 0:
                        new_pos = np.array([
                            current_target_info["pos"][0] + 4 +
                            ↪  idx_for_pos_calc,
                            current_target_info["pos"][1] + 1
                        ])
                    else:
                        new_pos = np.array([
                        current_target_info["pos"][0] + 4 + idx_for_pos_calc,
                        current_target_info["pos"][1]
                    ])
                else:
                    new_pos = np.array([
                        current_target_info["pos"][0] + 4 + 2 *
                        ↪  (idx_for_pos_calc - 1),
                        current_target_info["pos"][1]
                    ])
                already_exists = any(
                                    np.allclose(np.array(t["pos"]), new_pos)
                                    for t in self.targets
                                )
                if not already_exists:
                    self.targets.insert(1,{
                        "pos":new_pos,
                        "wait":[20,20],
                        "is_queue":True
                    })
                    self.chair_index += 1
                for t in self.targets:
                    if not isinstance(t["pos"], np.ndarray):
                        t["pos"] = np.array(t["pos"], dtype=float)
                self.target_index = max(min(self.target_index,
                ↪  len(self.targets) - 1), 0)
elif np.array_equal(initial_current_target_info["pos"], np.array([3.5, 4])):
↪  #wait_row_2
```

```
                    #・・・
                    #・・・
                    #・・・
                self.target_index = max(min(self.target_index,
                ↪  len(self.targets) - 1), 0)

```

本シミュレーションでは待ち行列が 1 列の場合と 2 列の場合による混雑状況の変化の検証を行う．このコードはシミュレーション設定で 1 列または 2 列のどちらかを選択し，それに基づいて待ち行列においてエージェントが立つべき座標を計算する処理をおこなっている．

待ち行列にエージェントを追加

initial __ current __ target __ info にエージェントの現在の目標地点の位置座標が格納されている．この initial __ current __ target __ info が各料理提供口の座標（行列の先頭座標）と一致するかの判定を行っている．座標が一致した行列の末尾にそのエージェントを追加する．そして，その行列の累計利用者数をインクリメントする．この累計利用者数と自分のエージェント ID を紐づけて辞書に保存する．これにより，自分が並んだ待ち行列において累計で何番目であるのかを記録することができる．

待機位置の計算

idx __ for __ pos __ calc はその待ち行列において自分は先頭から何番目に並んでいるかを表す．シミュレーション設定で 1 列または 2 列のどちらかを選択するかによって待ち行列におけるエージェントの待機座標の計算方法は大きく変わる．これらの待機座標の計算方法は第 6 章の待ち行列モデルで解説した数式に基づく．idx __ for __ pos __ calc を用いて現在自分が待ち行列において偶数番目に並んでいるのか，それとも奇数番目に並んでいるのかを判定している．エージェント間の距離は 2，1 列目と 2 列目の間の幅は 1 というようにそれぞれ定数で設定している．これらの計算によってエージェントの待ち行列における待機座標を求めることができる．

行動計画の更新

計算したエージェントの待ち行列における待機座標が目標地点が格納されているリストに含まれているかの判定を行う．これによって，同じ座標が何度も追加されることを防いでいる．そして，self.target.insert(1,...) で目標地点が格納されているリストのインデックス 1 番に座標を追加する．待ち行列の待機座標を追加する前の目標地点が格納されているリストのインデックス 0 番は食堂の入口の座標，そしてインデックス 1 番は料理提供口の座標となっていた．そのため，料理提供口の座標の前に，待ち行列における待機座標を追加するという処理を行っている．そして，このとき待機座標の他に待ち行列における待ち時間，その座標が待ち行列の一部であるかを判定するフラグといった情報も同時に追加している．待ち行列における待ち時間は計測を何度か行いそのデータに基づいている．self.chair __ index には座席の探索を開始する目標地点のインデックス番号が格納されているが，待ち行列における待機座標の情報が追加されたことによって，その分インデックス番号を一つ後ろにずらしている．

1 列固定の待ち行列

```
elif self._is_near(initial_current_target_info["pos"], [32, 7]): #wait_row_4
        if self not in wait_row_4:
            wait_row_4.append(self)
            wait_testrow.append(self.id)
            g_queue_counters["row_4"] += 1
            g_queue_agent_map["row_4"][self.id] = g_queue_counters["row_4"]
            # すでに登録済み → persistent_idx は固定
        persistent_idx = g_queue_agent_map["row_4"][self.id]
        my_list_index = wait_row_4.index(self)
        idx_for_pos_calc = my_list_index + 1
        new_pos = np.array([32 - idx_for_pos_calc, 7])
        new_target = {"pos": new_pos, "wait": [0.5, 0.5]}
        if persistent_idx not in row_target_4:
            row_target_4[persistent_idx] = new_target # 番号で場所を保存
        # --- ターゲットを self.targets に反映 ---
        for i in range(my_list_index):
            agent_in_front = wait_row_4[i]
            if agent_in_front.id in g_queue_agent_map["row_4"]:
                agent_in_front_persistent_idx =
                ↪  g_queue_agent_map["row_4"][agent_in_front.id]
                if agent_in_front_persistent_idx in row_target_4:
                    candidate_pos =
                    ↪  row_target_4[agent_in_front_persistent_idx]["pos"]
                    already_exists = any(
                        np.allclose(np.array(t["pos"]), candidate_pos)
                        for t in self.targets
                    )
                    if not already_exists and candidate_pos[0] <=
                    ↪  self.targets[self.target_index]["pos"][0]:
                        self.targets.insert(self.target_index, {
                            "pos": [candidate_pos[0], candidate_pos[1]],
                            "wait": [10.0, 10.0], #レジ下
                            "is_queue":True
                        })
                        self.chair_index += 1

        for t in self.targets:
            if not isinstance(t["pos"], np.ndarray):
                t["pos"] = np.array(t["pos"], dtype=float)
elif np.allclose(initial_current_target_info["pos"], np.array([32, 10])):
↪  #wait_row_5
```

```
                        #・・・
                        #・・・
                        #・・・
                t["pos"] = np.array(t["pos"], dtype=float)
```

1 列固定の待ち行列は，1 列または 2 列かを動的に切り替えることができる待ち行列と待ち行列における待機座標の求め方が異なるのみで，その他のロジックは同様である．また，1 列固定の待ち行列における待機座標の求め方も第 6 章の待ち行列モデルで解説した式に基づいている．wait __ row __ 4 と wait __ row __ 5 はそれぞれレジの座標を行列の先頭座標（x 座標が 32）と定義している．wait __ row __ 4 と wait __ row __ 5 はシミュレーション空間上において，x 座標がマイナス方向に待ち行列が伸長する．wait __ row __ 6 は待ち行列の先頭座標が 17，待ち行列の伸長方向はプラス方向と定義している．

一定人数を超過した際にずれて並ぶ待ち行列

```
elif np.allclose(initial_current_target_info["pos"], np.array([23.5, 18])) :  #wait_row_7
        # ウォーターサーバ待機登録
        if self not in wait_row_7:
            wait_row_7.append(self)
            g_queue_counters["row_7"] += 1
            g_queue_agent_map["row_7"][self.id] = g_queue_counters["row_7"]


        persistent_idx = g_queue_agent_map["row_7"][self.id]
        my_list_index = wait_row_7.index(self)
        idx_for_pos_calc = my_list_index + 1


        first_row_limit = 6  # Y=18 に並ぶ最大人数
        new_x = 23.5 + idx_for_pos_calc
        if idx_for_pos_calc <= first_row_limit:
            new_y = 18.0
        else:
            new_y = 17.0
        new_pos = np.array([new_x, new_y])


        new_target = {"pos": new_pos, "wait": [0.1, 0.1]}

        if persistent_idx not in row_target_7:
            row_target_7[persistent_idx] = new_target # 場所を保存
```

```
27
28
29          for i in range(my_list_index):
30              agent_in_front = wait_row_7[i]
31              if agent_in_front.id in g_queue_agent_map["row_7"]:
32                  agent_in_front_persistent_idx =
                    ↪  g_queue_agent_map["row_7"][agent_in_front.id]
33                  if agent_in_front_persistent_idx in row_target_7:
34                      candidate_pos =
                        ↪  row_target_7[agent_in_front_persistent_idx]["pos"]
35                      already_exists = any(
36                          np.allclose(np.array(t["pos"]), candidate_pos)
37                          for t in self.targets
38                      )
39                      if not already_exists and candidate_pos[0]
                        ↪  <self.targets[self.target_index]["pos"][0]:
40                          self.targets.insert(self.target_index, {
41                              "pos": [candidate_pos[0], candidate_pos[1]],
42                              "wait": [10.0, 10.0], #ウォーターサーバー
43                              "is_queue":True
44                          })
45
46          for t in self.targets:
47              if not isinstance(t["pos"], np.ndarray):
48                  t["pos"] = np.array(t["pos"], dtype=float)
```

一定人数を超過した際にずれて並ぶ待ち行列に関しても，今まで解説した待ち行列とは待機座標を求める計算方法が異なる．この待ち行列はウォーターサーバにおける待ち行列を表しており，この待ち行列が伸長した先には座席が存在しているため，直線的に待ち行列を伸ばし続けることは困難である．そのため，この待ち行列に並んでいる人数がある一定人数を超えたら，少し y 座標をずらして並び続けるというように閾値を設定し表している．この待ち行列では閾値を 6 人に設定し 7 人目以降はシミュレーション空間上で y 座標を少し下げた位置から再び並び続ける．

エージェントを待ち行列から削除する処理は以下のコードによって記述されている．

```
1       initial_current_target_info=current_target_info
2           new_pos = None
3           idx=None
4           if self in wait_row_1 and np.array_equal(current_target_info["pos"],
            ↪  np.array([3.5, 2.5])):
5                   wait_row_1.remove(self)
```

```
                # 辞書からも削除
                if self.id in g_queue_agent_map["row_1"]:
                    my_counter_num = g_queue_agent_map["row_1"][self.id]
                    if my_counter_num in row_target_1:
                        del row_target_1[my_counter_num]
                    del g_queue_agent_map["row_1"][self.id]
        elif self in wait_row_2 and
        ↪  np.array_equal(current_target_info["pos"], np.array([3.5, 5.5])):
        ↪  #wait_row_2
                    #・・・
                    #・・・
                    #・・・
        elif self in wait_row_7:
            if self.target_index > 0:
                previous_target_pos = self.targets[self.target_index -
                ↪  1]["pos"]

                if np.allclose(previous_target_pos, WATER_SERVER_POS):
                    wait_row_7.remove(self)
                    if self.id in g_queue_agent_map["row_7"]:
                        my_counter_num = g_queue_agent_map["row_7"][self.id]
                        if my_counter_num in row_target_7:
                            del row_target_7[my_counter_num]
                        del g_queue_agent_map["row_7"][self.id]
```

エージェントが待ち行列から離脱する処理は，エージェントが待ち行列での待ち状態から，サービスが開始したタイミングで行われる．wait __ row __ 1，wait __ row __ 2，wait __ row __ 3 ではエージェントが次に目指す目標地点として料理提供棚の座標が示されたタイミング．wait __ row __ 4，wait __ row __ 5 はレジの座標が示されたタイミング．wait __ row __ 6 は返却口の次の中継点の座標が示されたタイミングで待ち行列からの離脱処理を行っている．しかし，wait __ row __ 7 のウォーターサーバにおける待ち行列だけ扱いが少し異なっている．wait __ row __ 7 はウォーターサーバでのサービスが完了し次の目標地点へ向かう時，一つ前の目標地点がウォーターサーバの座標であったら待ち行列からの削除処理を行う．ウォータサーバだけは他の待ち行列とは異なりすべてのエージェントが必ず経由するようには設定されていない．また，ウォーターサーバを利用するタイミングも座席に座る前と後と複数パターンが存在している．つまり，ウォーターサーバを利用した後の中継点がエージェントによって異なることがある．そのため，ウォーターサーバにおける待ち行列は他の待ち行列とは違い，次の目標地点を示したタイミングではなく，ウォーターサーバの座標が一つ前の目標地点であると示されたときに待ち行列からの削除処理を行う．このように待ち行列での待ち状態から次の段階に進んだとみなされた場合に，待ち行列からの離脱処理が実行される．エージェントを待ち行列に加える際に，g __ queue __ agent __ map とエージェント ID を紐づけて辞書形式で記録していたがこの情報を削除する．これにより，辞書のサイズが肥大化するのを防いでいる．

（※文責: 島津諒雅）

A.4.4.3 Social Force と運動方程式

本シミュレーションにおけるエージェントの移動プロセスは Social Force Model に基づいている．そして Social Force Model によって算出された力を基に，エージェントの実際の位置座標や速度を運動方程式によって更新している．

Social Force Model

Social Force Model には第 6 章で解説したように以下の 3 つの力を考慮している．

- 目的地へと向かう希望推進力：F __ driving
- 壁や障害物を避けて通ろうとする心理的反発力：F __ rep
- エージェント同士で距離を取ろうとする心理的反発力：F __ social

目的地へと向かう希望推進力

```
    if self.is_waiting or self.reached_goal:
            return np.zeros(2)

        # ここから力の計算
        target_pos = self.active_target_pos

        # 目標に向かうための駆動力 (F_driving)
        desired_direction = target_pos - self.pos
        dist_to_target = np.linalg.norm(desired_direction)
        if dist_to_target > 1e-4: # ゼロ除算を避ける
            desired_direction /= dist_to_target

        is_final_goal = (self.target_index == len(self.targets) - 1)
        current_desired_speed = self.params['desired_speed']
        if is_final_goal:
            current_desired_speed *= self.params['exit_speed_multiplier']

        desired_velocity = current_desired_speed * desired_direction
        F_driving = (desired_velocity - self.velocity) * self.params['mass']
        ↪  / self.params['relaxation_time']
```

希望推進力を求めるうえで必要な変数には以下のものがある．

- エージェントの質量：self.params[‘mass’]
- エージェントの希望速度：self.velocity
- 目的地への単位ベクトル：desired __ direction
- エージェントの現在の速度ベクトル：current __ desired __ speed

- 緩和時間：self.params[‘relaxation __ time’]

ここでエージェントの質量と緩和時間，エージェントの希望速度はあらかじめ定義されているものを参照している．目的地への単位ベクトルは目標地点の位置座標と現在のエージェントの位置座標の差分を求め，そこからベクトルを計算する．そして正規化することによって，目的地への単位ベクトル（desired __ direction）を求めている．ここで，もしエージェントの次の目標地点であるとき，self.params[‘exit __ speed __ multiplier’]というものを定義している．これは，用事をすべて済ませて退出する際は，通常より早足になるという人間の心理を反映させたものである．これによって，希望推進力を求めるうえで必要な変数すべてを求めることができた．後は，これらの変数を用いて第 6 章の Social Force Model での数式に基づき計算をすることで希望推進力を求めることができる．

壁や障害物を避けて通ろうとする心理的反発力

```
    # 壁からの斥力 (F_rep)
        diffs_to_walls = self.walls - self.pos
        distances_to_walls = np.linalg.norm(diffs_to_walls, axis=1)
        in_range_mask = (distances_to_walls > 1e-4) & (distances_to_walls <
        ↪  self.params['rep_effective_range'])


        F_rep = np.zeros(2)
        if np.any(in_range_mask):
            effective_diffs = diffs_to_walls[in_range_mask]
            effective_dists = distances_to_walls[in_range_mask]
            force_vectors = self.params['rep_strength'] *
            ↪  np.exp(-effective_dists / self.params['rep_decay'])[:,
            ↪  np.newaxis] * (effective_diffs / effective_dists[:,
            ↪  np.newaxis])
            F_rep = -np.sum(force_vectors, axis=0)
```

壁や障害物からの反発力を求めるうえで必要な変数は以下の通りである．

- 壁の構成点の集合：self.walls
- 斥力の最大強度：self.params[‘rep __ strength’]
- 斥力の減衰距離：self.params[‘rep __ decay’]
- エージェントと壁の構成点との距離：distances __ to __ walls
- エージェントから壁へ向かう単位ベクトル：(effective __ diffs / effective __ dists)

ここで壁の構成点の集合，斥力の最大強度，斥力の減衰距離はあらかじめ定義されたものを参照している．はじめに，エージェントの現在の位置座標とすべての壁（self.pos）との距離を計算する．このときに，self.params[‘rep __ effective __ range’]でエージェントに影響を与える壁の有効範囲を定義している．これによって，有効範囲内のみの壁との距離のみを抽出している．これに

よって，壁や障害物からの反発力を求めるうえでの必要な変数はすべて求めることができた．後は，第 6 章の Social Force Model での壁からの反発力の式に基づき計算をする．先程の希望推進力とは異なり一つの力ではなく，有効範囲内のすべての壁からの反発力の和を求める必要がある．今までの計算であると，エージェントが壁に吸い寄せられる引力となっているため，最後にマイナスを掛け合わせることによって，壁や障害物からの反発力となる．

エージェント同士で距離を取ろうとする心理的反発力

```
    # 他のエージェントからの斥力 (F_social)
        F_social = np.zeros(2)

        # 自身が入場直後の無効化時間中の場合，他のエージェントからの斥力を無視する
        if self.time_since_active <
        ↪  self.params['collision_immunity_seconds']:
            pass
        else:
            for other in all_customers:
                if other.id == self.id or not other.is_active:
                    continue

                # 着席中のエージェントは斥力を発生させない（当たり判定を無効化）
                if other.is_seated:
                    continue

                # 入場直後の無効化時間の場合，そのエージェントからの斥力を無視する
                if other.time_since_active <
                ↪  self.params['collision_immunity_seconds']:
                    continue

                diff_to_other = self.pos - other.pos
                dist_to_other = np.linalg.norm(diff_to_other)
                if 0 < dist_to_other < self.params['social_force_range']:
                    force_magnitude = self.params['social_force_strength'] *
                    ↪  np.exp(-dist_to_other /
                    ↪  self.params['social_force_decay'])
                    force_direction = diff_to_other / dist_to_other
                    F_social += force_magnitude * force_direction
```

他のエージェントからの反発力を求めるうえで必要な変数は以下の通りである．

- 他のエージェントの集合：other
- 斥力の最大強度：self.params[‘social __ force __ strength’]

- 斥力の減衰距離：self.params[‘social __ force __ decay’]
- 他のエージェントとのユークリッド距離：dist __ to __ other
- 他のエージェントから自分へ向かう単位ベクトル：force __ direction

ここで他のエージェントの集合，斥力最大強度，斥力の減衰距離はあらかじめ定義されているものを参照している．はじめに，斥力の計算を行う対象のエージェントの判定をしている．対象外となるエージェントは自分自身，座席に座っているエージェント入場直後のエージェントがある．座席に座っているエージェントも対象に含めてしまうと，座席エリアの通行を妨げてしまうためそれを防ぐためである．入場直後のエージェントは，エージェントが出現直後は重なって生成される可能性があり他のエージェントとの反発力が大きくなりすぎてエージェントが弾け飛ぶことが危険性がある．そこで，self.params[‘collision __ immunity __ seconds’]でエージェントがシミュレーションに出現してから一定時間，反発力を無視するための時間を設定している．他エージェントからの反発力も壁からの反発力と同様に，self.params[‘social __ force __ range’]によって斥力計算を行う対象とする有効範囲の設定を行っている．まず，自分自身の位置座標と他のエージェントの位置座標とのベクトルと，距離を求めている．そして，２つのエージェント間のベクトルを距離で割ることによって，エージェント間の単位ベクトル（force __ direction）を求めることができる．これで，他のエージェントからの反発力を求めるうえで必要な変数がすべて求める事ができた．後は，第 6 章の Social Force Model での数式に基づき計算をすることによって，他のエージェントからの反発力を求めることができる．

以上のプロセスにより，「目的地へと向かう希望推進力」，「壁や障害物を避けて通ろうとする心理的反発力」，「エージェント同士で距離を取ろうとする心理的反発力」それぞれの力を求めることができた．

```
    F_driving + F_rep + F_social
```

これら 3 つの力の総和が最終的なエージェントの移動の制御をする力となる．

運動方程式

本シミュレーションでは，連続的な物理現象をコンピュータ上で扱うため，時間を微小なタイムステップで区切り，計算を行っている．エージェントの状態更新はニュートンの運動方程式に基づき行われている．

```
    # 運動方程式に基づいて速度と位置を更新
        dt = self.params['anim_interval'] / 1000.0  # 時間ステップを秒に変換
        # アクティブになってからの経過時間を更新
        if self.is_active and not self.is_waiting:
            self.time_since_active += dt

        # 加速度を計算 (a = F/m)
        acceleration = total_force / self.params['mass']

        # 速度を更新 (v = v0 + a*dt)
```

```
        self.velocity += acceleration * dt

        # 速度が速くなりすぎないように制限をかける
        is_final_goal = (self.target_index == len(self.targets) - 1)
        current_desired_speed = self.params['desired_speed'] *
        ↪  (self.params['exit_speed_multiplier'] if is_final_goal else 1.0)
        speed = np.linalg.norm(self.velocity)

        max_speed_mult = self.params['max_speed_multiplier']
        if speed > current_desired_speed * max_speed_mult:
             self.velocity = (self.velocity / speed) * (current_desired_speed
             ↪  * max_speed_mult

        # 位置を更新 (p = p0 + v*dt)
        self.pos += self.velocity * dt
```

加速度の算出

Social Force Model によって算出された合力（F __ total）とエージェントの質量（self.params[‘mass’]）を用いて，その瞬間のエージェントの加速度（acceleration）を求める．時間ステップΔ t（dt）は，1 フレームあたりの時間経過時間を表している self.params[‘anim __ interval’] を 1000 で割り，1 秒あたりの時間ステップを求めている．

速度の更新

現在のエージェントの速度（self.velocity）に加速度による変化量を加算し次ステップにおけるエージェントの速度を求めている．算出した加速度（acceleration）に時間ステップ（dt）を掛けたものを加算している．本シミュレーションにおいて，エージェント同士が過度に接近した場合などに反発力が大きくなりすぎて，速度が速くなりすぎる可能性がある．これを防ぐために，self.params[‘max __ speed __ multiplier’] で許容最大速度を設定し速度制限を行っている．

位置座標の更新

現在のエージェントの位置座標（self.pos）に速度（self.velocity）に時間ステップ（dt）を掛け合わせたものを加算することによって，次のステップにおけるエージェントの位置座標を算出している．

（※文責: 島津諒雅）

A.4.4.4　行動決定と状態更新

今まで述べてきた，Social Force Model や運動方程式は，エージェントが目的地へ向かうための物理的な移動メカニズムである．しかし，実際の食堂における利用者の行動を再現するためには，物理的な移動だけでなく，利用者の状況に応じた動的な意思決定も反映させる必要がある．そこで本章では，Customer クラスにおける「行動決定アルゴリズム」と「状態管理」について解説する．

これらの処理は apply __ movement __ and __ update __ state 関数で実装されている．具体的にこの関数では，「目標地点への到達判定」「動的なルート生成」，「状態管理」のロジックが記述されている．

目的地への到達判定・ターゲット更新

```
    def apply_movement_and_update_state(self, total_force):
        #・・・待機処理や物理計算・・・

        # 中継点への到達判定
        dist_to_target = np.linalg.norm(self.active_target_pos - self.pos)
        if dist_to_target < self.params['target_reach_threshold']:

            #・・・動的イベント処理・・・

            self.target_index += 1

            #・・・座席探索処理・・・

            self._set_next_target()



    def _set_next_target(self):
        #・・・A.4.4.2 待ち行列で解説した部分・・・

        base_pos = current_target_info2["pos"] # 待ち行列用の変更 current_target_info → current_target_info2
        is_special_target = (current_target_info2.get("is_chair", False) or # 待ち行列用の変更 current_target_info → current_target_info2
                             self.target_index == len(self.targets) - 1 or
                             np.array_equal(base_pos, WATER_SERVER_POS) or

                             current_target_info2.get("force_add_return_route",
                             False) or # 待ち行列用の変更
                             current_target_info → current_target_info2
                             current_target_info2.get("is_queue",False))


        if not is_special_target:
            random_radius = self.params['waypoint_randomness']
            angle = random.uniform(0, 2 * np.pi)
```

```
            radius = random.uniform(0, random_radius)
            offset = np.array([np.cos(angle), np.sin(angle)]) * radius
            self.active_target_pos = base_pos + offset
        else:
            self.active_target_pos = base_pos

```

シミュレーションは離散的な時間ステップで計算されるため，エージェントの座標が目標座標と完全に一致することが稀である．そのため，一定の距離内に入ったときに「目標地点へ到達」とみなす閾値を設定している．現在の目標地点の座標（self.active __ target __ pos）とエージェントの現在の位置座標（self.pos）との間のユークリッド距離を dist __ to __ target としている．この dist __ to __ target が閾値（self.params['target __ reach __ threshold']）を下回った場合，目標地点に到達したとみなし，到達後の処理が実行される．目標地点に到達したとみなされ到達判定が真となったとき，目標地点の情報が格納されているリスト（self.targets）の参照位置を進める．これにより，エージェントの目標地点が更新される．そして，__ set __ next __ target 関数を呼び出し，具体的な移動先の座標の計算を行っている．

現実世界において，人間は必ずしも経路上の特定の一点を厳密に通過するわけではない．人間一人一人には個人差があり，その通過位置は自然とずれが生じる．もし，すべてのエージェントが全く同じ座標を目指し移動した場合は，機械的で不自然な挙動となってしまう．そこで本シミュレーションでは，この人間特有の動きのばらつきを再現するために，中継点の座標決定においてランダム性を導入している．__ set __ next __ target 関数にはこの，中継点の座標のランダム性に関する具体的なロジックが記述されている．まず，目標地点にはランダム性を持たせるものと持たせないものの 2 つが存在している．座席，待ち行列での並ぶ位置座標，ウォーターサーバー，ゴールというようにこれらの地点においてはランダム性を持たせずに，完全に通過・到達するようにしたい．そこで，この特別な目標地点とそれ以外の目標地点の判定するためのフラグを is __ special __ target と定義した．もし，is __ special __ target が False であるとき，つまり特別な目標地点ではないときには目標地点の座標にランダム性を持たせなければならない．self.params['waypoint$_r$$andomness'$] を厳密な目標地点の座標とのずれの最大距離を表すパラメーターとして定義している．このパラメータと $random.uniform$ を用いることによって，ランダムな角度と半径を生成している．これらを曲座標変換することによって，オフセットベクトルを算出し，厳密な目標地点の座標に加算する．これによって，最終的なランダム性を持たせた目標地点の座標を決定している．一方，is __ $special$ __ $target$ が $True$ であるとき，つまり特別な目標地点であるときは，オフセットを加算せずに，あらかじめ定義されている目標地点の座標をそのまま目標座標として決定している．

動的な意思決定とルート生成

```
            # 待ち行列処理（動的な列選択）
            if is_queue_target:
                current_queue = None
                # ・・・どの列に並んでいるか特定・・・

```

```
                if current_queue is not None:
                    # 行列内の自分の順番を取得
                    current_idx = current_queue.index(self)

                    if current_idx >= 0:
                        # 順番に基づいて並ぶべき座標 (new_pos) を計算
                        # ・・・座標計算ロジック・・・

                        # ターゲットを動的に更新（列の最後尾を目指す）
                        self.active_target_pos = new_pos
                        self.targets[self.target_index]["pos"] = new_pos
                        return True # 移動を中断して整列

                    #・・・
                    #・・・
                    #・・・

            #着席後の行動分岐（ウォーターサーバーへの寄り道など）
            if is_chair_target and not self.post_seating_water_server_added:
                self.post_seating_water_server_added = True

                # 確率でウォーターサーバーに行くか決定
                if random.random() <
                ↪  self.params['post_seating_water_server_probability']:
                    # 現在のターゲット（座席情報）をコピーして復路用に保持
                    # ・・・

                    # ウォーターサーバー経由のルートをターゲットリストに挿入
                    self.targets.insert(insert_index, water_server_waypoint)
                    # ・・・
                    self.return_route_added = True

                    #・・・
                    #・・・
                    #・・・

            # 空席探索とルート生成（7番目の中継点通過後に実行）
            if self.target_index == self.chair_index and not
            ↪  self.chair_target_added:
                random_chair_pos = None

                # 空席リストから座席を一つ取得（確保）
```

```
                if AVAILABLE_CHAIRS:
                    random_chair_pos = AVAILABLE_CHAIRS.pop(0)
                    self.target_chair_pos = random_chair_pos

                if random_chair_pos is not None:
                    selected_pattern = None
                    chair_x, chair_y = random_chair_pos[0],
                    ↪  random_chair_pos[1]

                    for area in CHAIR_AREA_PATTERNS:
                        x_min, x_max = area['x_range']
                        y_min, y_max = area['y_range']
                        if (x_min <= chair_x <= x_max) and (y_min <= chair_y
                        ↪  <= y_max):
                            selected_pattern =
                            ↪  random.choice(area['patterns'])
                            break

                    new_waypoints = [CHAIR_WAYPOINTS[key] for key in
                    ↪  selected_pattern] if selected_pattern else []

                    # 滞在時間（食事時間）をランダムに決定
                    random_wait = random.uniform(MIN_CHAIR_WAIT_SECONDS,
                    ↪  MAX_CHAIR_WAIT_SECONDS)
                    chair_destination = {"pos": random_chair_pos, "wait":
                    ↪  random_wait, "is_chair": True}
                    new_waypoints.append(chair_destination)

                    # リストを更新
                    self.targets[self.target_index:self.target_index] =
                    ↪  new_waypoints

                    # 食事前のウォーターサーバー立ち寄り判定
                    if random.random() <
                    ↪  self.params['water_server_probability']:
                        # ウォーターサーバーを追加挿入
                        # ・・・

                else:
                    # 満席の場合（食事を諦めて退場）
                    self.targets = self.targets[:self.target_index]
```

```
79                      self.targets.append({"pos": GOAL_POS, "wait": [0, 0]})
80
81                  self.chair_target_added = True
```

本シミュレーションではエージェントが目標地点に到達した際，単純に次の目標地点へ進むだけでなく，その時点での空席状況や待ち行列の長さ，確率に基づいて，移動ルートを動的に変更するロジックを実装している．コード内には複数の分岐処理が存在しており，特に「空席探索」「待ち行列への整列」「確率的行動」の 3 点について解説する．

まず，エージェントの座席決定ロジックについて解説する．エージェントが座席エリア付近にある特定の中継点に到達した際，空いている座席を探し，目標地点として設定する．AVAILABLE __ CHAIRS は空席である座席のリストであり，このリストの先頭から座席座標を一つ取り出している．AVAILABLE __ CHAIRS はもともと座席の情報がランダムで格納されているため，先頭の座席座標を取り出しても空席の座席をランダムぜ選択していることになる．そして選択した座席の座標に応じて，最適な通路を経由するためのルートパターン（CHAIR __ AREA __ PATTERNS）を検索する．この検索によって得られた経路における中継点を目標地点の情報が格納されているリスト（self.targets）に追加している．もし，空席リストが空であった場合，エージェントの残りのルートをすべて破棄して出口に向かうように設定している．

次に，待ち行列の整列行動について解説する．レジや料理提供棚などの待ち行列が発生するエリアに到達した際，エージェントは列の最後尾を判断して並ぶ位置を決定している．待ち行列における待機座標の算出の仕方は A.4.4.2 で解説したとおりであり，毎フレーム計算をすることによって待ち行列における待機座標をリアルタイムに更新している．

最後に，ウォーターサーバーへの立ち寄り行動について解説する．ウォーターサーバーへの立ち寄りはすべての人が必ずするとは限らず，また立ち寄るタイミングも人によって様々である．self.params['water$_{s}erver_{p}robability$'] で，ウォーターサーバーへ立ち寄る確率を定義している．このパラメータに基づいて，ウォーターサーバーへ立ち寄るか判定を行う．この判定は，座席が決定したタイミング（食事前）と座席到着後（食事後）の 2 回行われている．食事前にウォーターサーバーへ立ち寄る場合は，ウォーターサーバーから座席へ向かうための経路における中継点の座標を目標地点の情報が格納されているリストに追加しする．そして，食事後にウォーターサーバーへ立ち寄る場合は，座席からウォーターサーバーへ向かうための経路における中継点，ウォーターサーバーから座席へ向かうための経路における中継点の座標を目標地点の情報が格納されているリストに追加する．このときの経路の検索はエージェントの座席決定ロジックで解説した方法と同様である．

待機時間の管理と座席リソースの排他制御

```
1       def apply_movement_and_update_state(self, total_force):
2           # ・・・ゴール判定など・・・
3
4           # 待機中の場合の処理
```

```
        if self.is_waiting:
            # タイマーの更新（カウントダウン）
            self.wait_timer -= 1

            # 待機時間が終了した場合
            if self.wait_timer <= 0:
                self.is_waiting = False # 待機フラグを下ろす
                self.is_seated = False  # 着席フラグを下ろす

                # ・・・座席リソースの解放（排他制御の解除）・・・
                # ・・・ウォーターサーバー等の一時離席ではなく、完全に席を立つ場合の処
                ↪ 理・・・

                if self.target_chair_pos is not None and self.has_left_chair
                ↪ == False:
                    # ・・・ウォーターサーバーへの一時離席かどうかの判定ロジック・・・

                    if is_final_wait_over and self.has_left_chair == False:
                        # 使用していた座席座標を共有リストに戻す
                        AVAILABLE_CHAIRS.append(self.target_chair_pos)
                        # 次の検索のためにリストをシャッフル
                        random.shuffle(AVAILABLE_CHAIRS)
                        self.has_left_chair = True

            # 待機中は物理的な移動を停止（速度を 0 にする）
            self.velocity = np.zeros(2, dtype=float)
            return True
```

エージェントが座席や料理提供棚に到達した際，物理的な移動を停止し，設定された時間だけその場に留まる「待機状態」への遷移が発生する．また，食事のためエージェントが着席している座席は他者が利用できないように占有し，離席時には再利用可能な状態に戻すという座席のリソース排他制御が必要である．

まず，待機状態の管理について解説する．目標地点情報に待機時間（wait）が設定されている場合，エージェントは移動モードから待機モードへと状態を遷移させる．エージェントの状態を管理するフラグ（is __ waiting）を True に変わるとエージェントは待機状態となり，Social Force Model による物理演算をスキップし，速度ベクトルをゼロベクトルに強制する．これにより，エージェントはその場に静止し続けるようになる．設定されている待機時間は，シミュレーションのフレームレートに基づいてフレーム数に変換され，wait __ timer に格納され，毎フレームこのタイマーを減算し，wait __ timer が 0 になった時点で待機終了と判断し，エージェントは待機モードから移動モードへと状態が遷移し移動を再開する．

次に，座席リソースの解放と排他制御について解説する．前述の座席決定ロジックの解説の中で，空席リスト（AVAILABLE __ CHAIRS）を用いて座席の座標を所得したと解説をした．ここでは，占有状態となった座席を解放状態にする際のロジックについて解説する．エージェントが座席に到達し，エージェントは待機状態となると同時に，座席は占有状態となる．そして，待機時間（wait __ timer）を減算していき，待機時間が 0 となり食事が終了した際に，自信が使用していた座席の座標を再び空席リスト（AVAILABLE __ CHAIRS）に追加することによって，座席は占有状態から解放状態へと遷移する．この時，即座に空席リスト（AVAILABLE __ CHAIRS）をシャッフルすることによって，次に空席リストから座席座標を参照するときにランダムな座席を選択されるようにする．

（※文責: 島津諒雅）

A.4.5　メイン関数

メイン処理とは、プログラムを実行したときに最初に動き始める中心部分であり、全体の流れを制御する役割を持つ処理である。本プログラムでは、入力データの読み込みや条件（レイアウト・行列など）の選択を行った後、シミュレーションを開始し、アニメーション更新処理を繰り返して人の移動や混雑状況を可視化する。さらに必要に応じて人口密度や衝突回数などの指標を計算し、結果の表示や評価に利用できるようになっている。

（※文責: 菊地皓太）

A.4.5.1　グローバル変数

```
# 人口密度グラフ用の変数を初期化（Lazy initialization のため None にしておく）
fig_density = None
ax_density = None
line_density = None

def run_simulation(selected_date_arg, df, layout_data,selected_queue_arg):
    global SELECTED_DATE,SELECTED_QUEUE_TYPE, SIMULATION_START_TIME  # グローバル変数を参照
    SELECTED_DATE = selected_date_arg  # 引数で受け取った日付を設定
    SELECTED_QUEUE_TYPE = selected_queue_arg

    # Customer クラスがグローバル参照する変数を設定
    global AVAILABLE_CHAIRS, GOAL_POS, INITIAL_POSITION, WATER_SERVER_POS  # 共通オブジェクト
    global WAYPOINTS_1, WAYPOINTS_2, WAYPOINTS_3  # 共通ルート
    global CHAIR_WAYPOINTS, CHAIR_AREA_PATTERNS, RETURN_ROUTE_PATTERNS, WATER_SERVER_RETURN_PATTERNS  # レイアウト別ルート

```

```
    # レイアウトデータの読み込み
    global REGISTER_POSITIONS, EXIT_POS, ENTRANCE_POS, RETURN_BOX_POS,
    ↪  WATER_SERVER_POS, TRASH_BOX_POS, TERRACE_EXIT_POS
    global WALLS_LINE, SPACES_RECT, TABLES, CHAIR_POSITIONS

    REGISTER_POSITIONS = layout_data['REGISTER_POSITIONS']
    EXIT_POS = layout_data['EXIT_POS']
    ENTRANCE_POS = layout_data['ENTRANCE_POS']
    RETURN_BOX_POS = layout_data['RETURN_BOX_POS']
    WATER_SERVER_POS = layout_data['WATER_SERVER_POS']
    TRASH_BOX_POS = layout_data['TRASH_BOX_POS']
    TERRACE_EXIT_POS = layout_data['TERRACE_EXIT_POS']

    WALLS_LINE = layout_data['WALLS_LINE']
    SPACES_RECT = layout_data['SPACES_RECT']

    TABLES = layout_data['TABLES']
    CHAIR_POSITIONS = layout_data['CHAIR_POSITIONS']

    WAYPOINTS_1 = layout_data['WAYPOINTS_1']
    WAYPOINTS_2 = layout_data['WAYPOINTS_2']
    WAYPOINTS_3 = layout_data['WAYPOINTS_3']

    CHAIR_WAYPOINTS = layout_data['CHAIR_WAYPOINTS']
    CHAIR_AREA_PATTERNS = layout_data['CHAIR_AREA_PATTERNS']
    RETURN_ROUTE_PATTERNS = layout_data['RETURN_ROUTE_PATTERNS']
    WATER_SERVER_RETURN_PATTERNS =
    ↪  layout_data['WATER_SERVER_RETURN_PATTERNS']

    AVAILABLE_CHAIRS = layout_data['AVAILABLE_CHAIRS']
    GOAL_POS = EXIT_POS
    INITIAL_POSITION = ENTRANCE_POS
```

本プログラムでは，シミュレーション全体で共通して参照する情報を，グローバル変数として管理している．

まず，人口密度グラフ表示機能に用いる変数として，fig __ density，ax __ density，line __ density を None で初期化している．これにより，人口密度グラフを常時生成するのではなく，必要になった時点で初めて作成する（Lazy initialization）設計となっており，不要な描画処理やウィンドウ生成を抑えることができる．

fig __ density：人口密度グラフ用の Figure（別ウィンドウ）

ax __ density：そのグラフの Axes（描画領域）

line __ density：折れ線（Line2D）の実体

次に，run __ simulation() 関数では，GUI で選択された日付や待ち行列タイプの情報を引数として受け取り，それらを SELECTED __ DATE および SELECTED __ QUEUE __ TYPE に代入することで，シミュレーションの実行条件をグローバルに確定させている．これにより，後続の処理（アニメーション更新処理や Customer クラス内部）からも，選択された実行条件を参照できるようになる．
SELECTED __ DATE：GUI で選んだ日付
SELECTED __ QUEUE __ TYPE：GUI で選んだ行列タイプ

run __ simulation() 内では，Customer クラスが行動決定や経路選択を行う際に必要となる情報を，グローバル変数へ展開している．具体的には，椅子の候補リスト（AVAILABLE __ CHAIRS），最終目標地点（GOAL __ POS），初期出現位置（INITIAL __ POSITION），給水機位置（WATER __ SERVER __ POS）など，全エージェントで共通となる環境設定を設定している．また，WAYPOINTS __ 1，WAYPOINTS __ 2，WAYPOINTS __ 3 により基本的な移動経路（中継点列）を定義し，加えて椅子周辺への移動経路や返却口へ向かう経路など，レイアウトに依存する行動パターンを CHAIR __ WAYPOINTS や RETURN __ ROUTE __ PATTERNS 等の変数に格納している．

レイアウト情報の読み込みについては，layout __ data という辞書型データから，各種座標情報を取得してグローバル変数に代入している．
REGISTER __ POSITIONS：レジ
EXIT __ POS：退出位置
ENTRANCE __ POS：入口位置
RETURN __ BOX __ POS：返却口
WATER __ SERVER __ POS：ウォーターサーバー
TRASH __ BOX __ POS：ゴミ箱
TERRACE __ EXIT __ POS：テラスへの出口

同様に，壁や通行不可領域を示す WALLS __ LINE，SPACES __ RECT，および机や椅子の配置情報（TABLES，CHAIR __ POSITIONS）も展開することで，エージェントが障害物を回避しながら移動できる環境を整えている．
WALLS __ LINE：壁（線分障害物）
SPACES __ RECT：入れない領域（矩形）
TABLES：机の位置情報（障害物 / 座席の周辺）
CHAIR __ POSITIONS：椅子の座標

最後に，GOAL __ POS を EXIT __ POS に，INITIAL __ POSITION を ENTRANCE __ POS に設定することで，「入口から出現して出口を最終目的地とする」というシミュレーションの基本構造を確定している．以上により，本ブロックはシミュレーション実行前に環境を初期化し，レイアウト条件に応じた共通パラメータを全体へ共有する役割を担っている．

（※文責: 菊地皓太）

A.4.5.2 アニメーション処理

```
def init():
    """アニメーションを初期化する"""
    for p in agent_patches: p.set_visible(False)
    for patch in target_chair_patches: patch.set_visible(False)
    return agent_patches + target_chair_patches

def update(ignored_frame):
    """アニメーションの各フレームを更新する"""

    nonlocal g_current_frame, customers, total_collision_count # AVAILABLE_CHAIRS はグローバル変数のため nonlocal から削除
    global AVAILABLE_CHAIRS # グローバル変数として認識させる

    if not animation_state['is_running']:
        return agent_patches + target_chair_patches

    frame = g_current_frame

    # タイトル・日時・仮想時間を表示
    elapsed_virtual_minutes = frame / frames_per_virtual_minute
    current_virtual_time = start_time_dt + timedelta(minutes=elapsed_virtual_minutes)
    time_str = current_virtual_time.strftime('%H:%M:%S')
    ax.set_title(f"食堂人流シミュレータ （{SELECTED_DATE}） 仮想時間: {time_str}")

    # 全エージェントの次の移動ベクトルを計算する
    forces = []
    active_customers_list = [c for c in customers if c.is_active]
    for customer in customers:
        if customer.is_active:
            force = customer.calculate_forces(active_customers_list)
            forces.append(force)
        else:
            forces.append(np.zeros(2))

    # 計算されたベクトルを適用して全エージェントの位置を更新
    all_finished = True
    remove_customers=[]
```

```
    for i, customer in enumerate(customers):
        if not customer.is_active and frame < customer.entry_frame:
            all_finished = False
            continue

        if not customer.is_active:
            customer.is_active = True
            agent_patches[customer.id].set_color(customer.color)
            agent_patches[customer.id].set_visible(True)

        moving = customer.apply_movement_and_update_state(forces[i])

        if moving:
            all_finished = False
            pos = customer.pos
            agent_patches[customer.id].center = (pos[0], pos[1])

    g_current_frame += 1
    return agent_patches + target_chair_patches + [collision_scatter,
    ↪  collision_text]

# アニメーションの定義（通常時）
ani = FuncAnimation(fig, update, frames=None,
                    init_func=init, interval=ANIMATION_INTERVAL, blit=False,
                    ↪  cache_frame_data=False)
```

本プログラムでは，食堂内における人流の様子を，時間経過とともに連続的に可視化するために，Matplotlib のアニメーション機能である FuncAnimation を用いて描画更新を行っている．アニメーション処理は「初期化（init）」と「フレーム更新（update）」の二種類の関数を中心に構成されており，FuncAnimation はこれらを適切な順序で呼び出すことで，画面表示を更新し続けるメインループとして機能する．

FuncAnimation によるアニメーションの基本構造

Matplotlib の FuncAnimation は，単に図を描画するのではなく，指定した更新関数を一定間隔で繰り返し呼び出すことで，図中の要素（線、点、パッチなど）の状態を連続的に変化させる仕組みである．一般的に FuncAnimation(fig, update, ...) の形で生成され，与えられた Figure 上で更新処理を繰り返し実行する．これにより，シミュレーションの各ステップ（各フレーム）における状態を画面に反映できる．

本プログラムにおける FuncAnimation は，開始時に init __ func として設定された初期化関数 init() を 1 回呼び出し，その後 interval=ANIMATION __ INTERVAL で指定されたミリ秒間隔ごとに更新関数 update() を呼び出す構造になっている．この更新関数の反復実行が，アニメーション表示における「メインループ」に相当する．

init() による初期状態の整備と描画の整合性確保

init() はアニメーション開始時の描画状態を統一するための初期化関数である．ここで重要なのは，アニメーションの初期状態が曖昧であると，初回の描画時に意図しない表示（本来出現していない客が見える，椅子ターゲットが残っている，前回の更新が残るなど）が発生する可能性がある点である．そのため本プログラムでは，初期化の段階で描画オブジェクトを明示的に初期状態へ戻す処理が行われている．

具体的には，客エージェントを表す円のパッチ群（agent __ patches）をすべて visible=False に設定し，さらに椅子ターゲットを表示するためのパッチ群（target __ chair __ patches）も同様に非表示化している．これにより，アニメーション開始直後の状態が「客がまだ登場していない（または表示対象ではない)」状態として統一され，シミュレーションの開始が視覚的にも明確になる．

また，init() の最後でこれら描画オブジェクト群を返却している点も重要である．FuncAnimation は「どの要素を更新対象として扱えばよいか」を知る必要があるため，更新対象の Artist（Matplotlib の描画要素）を返すことが推奨されている．これにより Matplotlib 側は，どの要素を描き直すべきかを認識でき，描画更新が適切に管理される．特に描画要素が多数存在する場合，更新対象を返すことは描画整合性の確保だけでなく，処理負荷の削減にもつながる．

update() によるフレームごとの状態更新と描画の同期

update() はアニメーションの各フレームで繰り返し実行される更新関数であり，プログラム全体の中でも最も重要な処理が集中している部分である．FuncAnimation により一定間隔で呼び出されるため，update() 内で「1 フレーム＝ 1 ステップ」としてシミュレーションを進めながら描画を更新する構造となる．

再生・停止制御

update() の冒頭では，animation __ state['is __ running'] の状態を参照している．これは，ユーザが UI（再生/停止ボタン）を操作した場合に，アニメーション更新処理を停止できるようにするための仕組みである．停止状態のときは update() が呼び出されても内部処理を実行せず，描画要素を返して終了することで，表示状態を保ったまま時間の進行のみを止めることができる．これは，実用的なシミュレーションツールとして操作性を高める重要な設計である．

フレーム管理と仮想時間表示

シミュレーションにおける現在フレーム番号は g __ current __ frame により管理されている．update() では frame = g __ current __ frame として現在フレームを取得し，処理の最後で g __ current __ frame += 1 を行うことで次フレームへ進めている．このようにフレーム番号を明示的に管理することで，時間ジャンプ機能など他の UI 処理と連携しやすい利点がある．

また，フレーム数から経過仮想時間（virtual time）を算出し，start __ time __ dt に加算することで現在の仮想時刻を計算している．計算された時刻を文字列化し，ax.set __ title() により図のタイトルとして表示することで，利用者は「現在のシミュレーションが何時相当なのか」を視覚的に把握できる．これは特に混雑が起きる時刻帯を比較・分析する際に有効であり，データと可視化の対応関係を明確にする効果がある．

entry __ frame による時間差入場と可視化

各エージェントは entry __ frame という値を保持しており，これが「何フレーム目からシミュレーション上に登場するか」を表している．update() 内では，まだアクティブでないエージェントについて frame ¡ entry __ frame の間は処理をスキップし，指定フレームに到達した時点で customer.is __ active = True として有効化し，同時に該当する円パッチを set __ visible(True) にすることで画面上に表示する．

この処理により，全員が一斉に登場するのではなく，現実の人流に近い形で「時間差で入場する」という挙動が表現される．また，パッチに色を設定する処理もここで行われており，各エージェントを視覚的に区別する仕組みとして機能する．

移動と状態遷移

アクティブなエージェントに対しては，あらかじめ計算した forces[i] を入力として apply __ movement __ and __ update __ state() を呼び出す．ここでは単なる座標更新だけでなく，目的地の切り替え，待機状態，行列参加，着席・退席，返却行動など，エージェントの状態遷移も含めた中心的な処理が実行されると考えられる．つまり，update() は「全体の駆動役」であり，行動ルールの詳細は Customer クラス側へ委譲する構造となっている。

移動が実際に発生した場合には，agent __ patches[customer.id].center = (pos[0], pos[1]) により円パッチの描画位置を更新する．これにより内部状態（customer.pos）と外部表示（円パッチ）が同期され，画面上でエージェントが動いて見える．

描画要素の返却

update() の最後でも init() と同様に描画オブジェクト群を返却している．これにより Matplotlib は「このフレームで更新された要素」を把握できる．本プログラムでは，エージェントや椅子表示に加えて，衝突表示用の collision __ scatter や collision __ text なども戻り値に含めており，エージェント可視化だけでなく補助指標（衝突位置や回数など）も同一の更新サイクルの中で描画管理している点が特徴である．

（※文責: 菊地皓太）

A.4.5.3　読み込み

```
# CSV ファイルの読み込み
if __name__ == '__main__':
    try:
        df_main = pd.read_csv('Count.csv') #'Count.csv'
    except FileNotFoundError:
        raise FileNotFoundError("Count.csv ファイルが見つからない")

# CSV ヘッダから日付候補を取得
dates = df_main.columns[1:].tolist() + ['テスト用データ']

# 選択されたレイアウト名からレイアウトデータを読み込む
layout_data = layout.get_layout(selected_layout)
```

本プログラムの起動時メイン処理（if ____ name ____ == ’ ____ main ____’:）では，シミュレーションを開始する前段階として，外部データの取得（読み込み）を行い，ユーザが条件を選択できるように準備を整えている．特に重要なのは，人の入場数を与える Count.csv の読み込み，その CSV に含まれる日付情報の抽出，選択されたレイアウトに対応する環境データの取得，という 3 種類の読み込み処理である．

まず，シミュレーションの基礎となる人数データを取得するために pd.read __ csv(’Count.csv’) を実行し，CSV ファイルを df __ main（DataFrame）として読み込む処理を行っている．

この CSV は，時刻帯ごとの入場人数など，シミュレーションにおいて「いつ・どれだけの客が発生するか」を決定する情報源であり，プログラムの入力データとして最も基本的な役割を担う．加えて，読み込み処理は try-except により例外処理が組み込まれており，Count.csv が存在しない場合には FileNotFoundError を発生させて明示的に処理を停止する構造となっている．

これにより，入力データが欠落したまま曖昧な状態でシミュレーションを進めてしまうことを防ぎ，エラー原因を利用者に分かりやすく提示できる．

次に，CSV の内容そのものをすべて解析する前段階として，DataFrame の列名から日付候補を抽出する処理が行われる．具体的には，df __ main.columns[1:] により「2 列目以降の列名」を取り出し，これを日付リスト（dates）として利用する．

ここで 1 列目を除外している理由は，一般に CSV の先頭列が「時刻」等の基準列であり，2 列目以降に「各日付の人数データ」が配置される形式を想定しているためである．また，この日付候補リストの末尾に [’ テスト用データ’] を追加している点も重要である．これは，実データに依存しない動作を可能にするための配慮であり，開発・検証段階での運用性を高める工夫といえる．

さらに，ユーザが UI でレイアウトを選択した後，その選択結果に応じて環境データを取得する処理が実行される．本プログラムでは layout.get __ layout(selected __ layout) を呼び出すことで，選択されたレイアウト名に対応するレイアウト情報（入口・出口・レジ位置，壁，机，椅子，移動経路などの座標群）を layout __ data として取得している．

これは「レイアウトの読み込み」に相当し，以降のシミュレーション本体（run __ simulation）が，同じ人数データであっても異なる空間配置条件のもとで動作するための基盤となる．つまり，人数データ（Count.csv）という需要側の読み込みに対して，レイアウトデータは環境（供給側）条件の読み込みであり，両者が揃うことで初めてシミュレーションの前提条件が確定する．

以上より，起動時のメイン処理における「読み込み」は，Count.csv を読み込み，シミュレーションに必要な入力データを確保すること，CSV の列名から日付候補を抽出してユーザに選択させること，選択されたレイアウトに応じた環境データを取得してシミュレーション条件を確定させること，の三段階で構成されている．これらの処理によって，ユーザが選択した条件に基づき，後続の本編処理（run __ simulation およびアニメーション更新）へ安全に移行できるよう設計されている．

（※文責: 菊地皓太）

A.4.5.4　各 UI とウィンドウ表示

```
# 選択用のウィンドウとウィジェットを作成
fig_selector, ax_selector = plt.subplots(figsize=(6, 8))

```

```
# レイアウト選択 UI
ax_selector.axis('off')
ax_radio_layout = plt.axes([0.1, 0.88, 0.8, 0.08])
ax_radio_layout.set_title('1. レイアウトを選択')
radio_layouts = RadioButtons(ax_radio_layout, layouts)

# 待ち行列選択 UI
ax_radio_queue = plt.axes([0.1, 0.76, 0.8, 0.08])
ax_radio_queue.set_title('2. 待ち行列を選択')
radio_queue = RadioButtons(ax_radio_queue, queue_types)

# 日付選択 UI
ax_radio_date = plt.axes([0.1, 0.12, 0.8, 0.60])
ax_radio_date.set_title('3. 日付を選択')
radio_dates = RadioButtons(ax_radio_date, dates)

# シミュレーション開始ボタン
ax_start_button = plt.axes([0.35, 0.03, 0.3, 0.06])
btn_start = Button(ax_start_button, '開始')

def start_simulation(event):
    """開始ボタンが押されたときの処理"""
    selected_layout = radio_layouts.value_selected
    selected_queue = radio_queue.value_selected
    selected_date = radio_dates.value_selected

    layout_data = layout.get_layout(selected_layout)

    plt.close(fig_selector) # 選択ウィンドウを閉じる

    run_simulation(selected_date, df_main, layout_data,selected_queue)

btn_start.on_clicked(start_simulation)
plt.show()

# シミュレーション本体ウィンドウ
fig, ax = plt.subplots(figsize=(16, 7)) # 横 16 インチ、縦 7 インチ
fig.subplots_adjust(right=0.82, bottom=0.15) # UI ウィジェット用のスペースを確保
ax.set_xlim(0, 70)
ax.set_ylim(0, 30)
ax.set_aspect("equal") # 縦横比を 1:1 にする
ax.grid(True, linestyle="--", alpha=0.3)
```

```
46
47  # 範囲選択 UI と人口密度グラフ用の別ウィンドウ表示
48  # ウィンドウが存在しない、または閉じられている場合は作成する
49  if fig_density is None or not plt.fignum_exists(fig_density.number):
50      fig_density, ax_density = plt.subplots(figsize=(6, 4))
51      fig_density.canvas.manager.set_window_title('選択範囲の人口密度推移')
52      line_density, = ax_density.plot([], [], 'b-', marker='o', markersize=3)
53      # 初回作成時は表示
54      fig_density.show()
55
56  # 左クリックドラッグで範囲選択、範囲は赤枠で表示
57  rect_selector = RectangleSelector(
58      ax, on_select_region,
59      useblit=True,
60      button=[1],
61      minspanx=1, minspany=1,
62      spancoords='data',
63      interactive=True,
64      props=dict(facecolor='red', edgecolor='red', alpha=0.2, fill=True)
65  )
66  ax.rect_selector = rect_selector
67
68  # 時刻ジャンプ用チェックボックス UI
69  # 時刻選択用の CheckButtons を配置する描画領域を作成
70  ax_check_jump = fig.add_axes([0.83, 0.05, 0.15, 0.25],
    ↪  facecolor='lightgoldenrodyellow')
71
72  # CheckButtons ウィジェットを作成
73  check_jump = CheckButtons(ax_check_jump, time_labels_for_check,
    ↪  actives=[False] * len(time_labels_for_check))
74
75  def on_time_select_check(label):
76      """チェックボックスで時刻が選択されたときに呼び出される関数"""
77      ...
78
79  # CheckButtons のクリックイベントに上で定義した関数を接続
80  check_jump.on_clicked(on_time_select_check)
81
82  plt.show()
```

本プログラムでは，利用者がシミュレーション条件を選択し，その結果に基づいてシミュレーション本編のアニメーションを実行・操作できるようにするため，Matplotlib が提供するウィ

ジェット機能（RadioButtons，Button，CheckButtons，RectangleSelector など）を用いて UI を構築している．UI は大きく分けて，起動直後に表示される「条件選択ウィンドウ」，シミュレーション実行中に表示される「本編ウィンドウ上の操作 UI」，操作に応じて追加表示される「人口密度推移の別ウィンドウ」，の 3 種類で構成されている．

まず起動時には，シミュレーション条件（レイアウト，待ち行列形式，日付）を選択するための専用ウィンドウを生成する．ここでは plt.subplots(figsize=(6, 8)) により fig __ selector（ウィンドウ全体）と ax __ selector（描画領域）を作成し，ax __ selector.axis('off') により座標軸を非表示化することで，グラフ描画というより「UI パネル」として使える画面を用意している．その上で，plt.axes([x, y, w, h]) を用いてウィンドウ内の複数位置に小さな描画領域を確保し，そこへ RadioButtons を配置することで，レイアウト選択・待ち行列選択・日付選択の 3 種類のラジオボタン UI を縦方向に並べている．この配置方法は，Matplotlib の「図（Figure）内に複数の Axes を置く」仕組みをそのまま UI レイアウトとして転用したものであり，ウィジェットの表示位置や大きさを数値で直接制御できる利点がある．一方で，座標は相対値で指定されるため，UI の見た目を整えるためには figsize や各 plt.axes(...) の値を調整する必要がある，という特性も併せ持つ．

次に，条件が選択された後にシミュレーションを開始するための「開始ボタン」を作成している．ボタンは Button(ax __ start __ button, ' 開始') で生成され，btn __ start.on __ clicked(start __ simulation) によりクリック時のコールバック関数を紐付けている．これにより，利用者がボタンを押すと start __ simulation が呼び出され，ラジオボタンの現在値（value __ selected）から各選択結果を取得する．そして，条件が確定した時点で plt.close(fig __ selector) により選択ウィンドウを閉じ，続けて run __ simulation(...) を呼び出すことで本編へ処理が移行する．ここで plt.show() が呼ばれていることは重要であり，Matplotlib のイベントループを開始して実際にウィンドウを表示し，ユーザ操作（クリック等）を受け付ける入口となっている．つまり，この段階では「条件選択ウィンドウを表示して操作待ちする」こと自体が起動時メイン処理の主要な役割となっている．

続いてシミュレーション本編では，まず fig, ax = plt.subplots(figsize=(16, 7)) により広い描画領域を持つウィンドウを作成し，fig.subplots __ adjust(right=0.82, bottom=0.15) によって右側および下側に余白を確保している．この余白は，アニメーション表示そのもの（食堂レイアウトとエージェント）を邪魔しない位置に UI ウィジェットを追加配置するためのスペースであり，単に見た目を整えるだけでなく「本編表示と操作 UI を同一ウィンドウに共存させる」ための設計上の重要な前処理である．さらに座標範囲（set __ xlim, set __ ylim）やアスペクト比（set __ aspect("equal")）を固定することで，食堂の平面図が歪まず，移動距離や混雑密度の視覚的比較が行いやすい状態に整えている．

本編ウィンドウ上での操作 UI として代表的なのが，時刻ジャンプ用の CheckButtons である．これは fig.add __ axes([0.83, 0.05, 0.15, 0.25]) により，先に確保した右側余白領域にチェックボックス専用の Axes を作り，そこへ CheckButtons(...) を配置して実現している．チェックボックスのクリックイベントには check __ jump.on __ clicked(on __ time __ select __ check) によりコールバックが接続され，ユーザが特定時刻を選択すると，その時刻へシミュレーション状態（フレーム）を移動させる処理へつながる構造になっている．このように，本編ウィンドウは「表示するだけの画面」ではなく，ユーザ操作を受け取ってシミュレーション状態を変化させる操作盤として機能する．

さらに，本プログラムは「範囲選択による分析」を行えるように，RectangleSelector を用いたドラッグ選択 UI も提供している．これは RectangleSelector(ax, on __ select __ region, ...) により，メイン描画 Axes（ax 上で左クリックドラッグをすると矩形範囲を選べるようにしたものであ

る．ここで選択が確定すると on __ select __ region が呼ばれ，その内部で人口密度推移表示用の別ウィンドウ（fig __ density）を必要に応じて新規作成している．特に fig __ density が未作成，または一度閉じられて存在しない場合には plt.subplots(figsize=(6, 4)) で新しい Figure を生成し，ウィンドウタイトルを設定した上で，fig __ density.show() により追加ウィンドウを表示する．この処理により，シミュレーション本編の表示を保ったまま，別ウィンドウに分析結果（人口密度の推移）を表示できる構成となっており，可視化と分析を並行して行える点が特徴である．

最後に，これらの UI が実際に画面に表示され，クリックや選択といった操作を受け付けられるようになるためには，Matplotlib のイベントループを起動する plt.show() が必要である．本プログラムでは，起動時の条件選択画面の表示（plt.show()）と、本編ウィンドウの表示（plt.show()）がそれぞれ適切なタイミングで呼ばれており，前者は「選択が完了するまで待つ」，後者は「シミュレーションを動かしながら操作を受け付ける」という役割分担になっている．

以上のように，本プログラムの UI・ウィンドウ表示処理は，起動時の条件選択 UI，本編ウィンドウ上の操作 UI，分析用の追加ウィンドウ表示を組み合わせることで，利用者が条件設定・実行・操作・分析を一連の流れとして行えるように設計されている．

（※文責: 菊地皓太）

A.4.5.5　人口密度

```python
def population_density(customers, a, b, c, d): # 人口密度の表示

    if len(customers) == 0:
        return 0.0

    x_min, x_max = min(a, b), max(a, b)
    y_min, y_max = min(c, d), max(c, d)

    population = 0
    for person in customers:
        if not person.is_active:
            continue

        x, y = person.pos[0], person.pos[1]
        if x_min <= x <= x_max and y_min <= y <= y_max:
            population += 1

    # シミュレーション上の面積を計算
    sim_area = abs((x_max - x_min) * (y_max - y_min))

    real_area = sim_area * 0.25

    return population / real_area if real_area != 0 else 0.0
```

```
24
25  # 人口密度グラフ表示用の関数定義
26  # グラフ用データ保持変数
27  density_history = []      # 密度の履歴
28  selected_region = None    # 選択された範囲
29
30  def on_select_region(eclick, erelease):
31      """範囲選択が完了したときに呼ばれる"""
32      nonlocal selected_region, density_history
33      # グローバルスコープの変数を参照・更新
34      global fig_density, ax_density, line_density
35
36      # None チェック（手動設定の場合があるため）
37      if eclick is not None and erelease is not None:
38          x1, y1 = eclick.xdata, eclick.ydata
39          x2, y2 = erelease.xdata, erelease.ydata
40          # 範囲を保存
41          selected_region = (x1, x2, y1, y2)
42
43      # ウィンドウが存在しない、または閉じられている場合は作成する
44      if fig_density is None or not plt.fignum_exists(fig_density.number):
45          fig_density, ax_density = plt.subplots(figsize=(6, 4))
46          fig_density.canvas.manager.set_window_title('選択範囲の人口密度推移')
47          line_density, = ax_density.plot([], [], 'b-', marker='o',
            ↪  markersize=3)
48          # 初回作成時は表示
49          fig_density.show()
50      else:
51          # 既に存在する場合はクリアして再利用
52          ax_density.cla()
53          # 線オブジェクトを再取得
54          line_density, = ax_density.plot([], [], 'b-', marker='o',
            ↪  markersize=3)
55
56      # データをリセットしてグラフ設定
57      density_history.clear()
58      ax_density.grid(True)
59
60      if selected_region:
61          ax_density.set_title(
62              f"人口密度推移 : x=[{selected_region[0]:.1f},
                ↪  {selected_region[1]:.1f}], "
```

```
            f"y=[{selected_region[2]:.1f}, {selected_region[3]:.1f}]"
        )

    # 横軸のラベル
    ax_density.set_xlabel("時間")

    # 縦軸のラベル
    ax_density.set_ylabel("人口密度 (人/m^2)")

    # 描画更新
    fig_density.canvas.draw()
    print(f"範囲選択: {selected_region}")

# 左クリックドラッグで範囲選択、範囲は赤枠で表示
rect_selector = RectangleSelector(
    ax, on_select_region,
    useblit=True,
    button=[1],
    minspanx=1, minspany=1,
    spancoords='data',
    interactive=True,
    props=dict(facecolor='red', edgecolor='red', alpha=0.2, fill=True)
)
ax.rect_selector = rect_selector

UPDATE_FREQUENCY = 44 # 仮想時間で約 15 秒ごとに人口密度更新

# fig_density が None でなく、かつウィンドウが開いている (番号が存在する) 場合のみ更新
if fig_density is not None and plt.fignum_exists(fig_density.number) and
↪   selected_region is not None and frame % UPDATE_FREQUENCY == 0:
    # 選択範囲の密度を計算
    d = population_density(customers,
                           selected_region[0], selected_region[1],
                           selected_region[2], selected_region[3])

    # データを追加
    density_history.append(d)

    # グラフのデータ更新
    x_data = range(len(density_history))
    line_density.set_data(x_data, density_history)

```

```
        # 軸の自動調整
        ax_density.relim()
        ax_density.autoscale_view()

        # 描画反映
        fig_density.canvas.draw_idle()

# --- テスト用データ選択時の特別処理：人口密度グラフを自動表示 ---
# 範囲を固定 (24.0, 50.0, 0.0, 6.0)
selected_region = (24.0, 50.0, 0.0, 6.0)
# on_select_region を呼び出してウィンドウを作成 (引数は None で内部処理させる)
on_select_region(None, None)
# 矩形選択ツールにもこの範囲を反映させる（視覚的フィードバック用）
rect_selector.extents = (24.0, 50.0, 0.0, 6.0)
```

本プログラムでは，食堂内の混雑状況を定量的に把握するための機能として，「任意に選択した範囲内の人口密度（人／ m^2）の時間推移」を計算し，別ウィンドウの折れ線グラフとして表示する仕組みを実装している．本機能は，単にエージェントの動きを眺めるだけでは把握しづらい「特定地点（レジ前や返却口付近など）の混雑がいつ強くなるか」「混雑が継続する時間がどれほどか」といった傾向を，数値として比較・評価できるようにすることを目的としている．

まず人口密度の算出は，専用関数 population __ density(customers, a, b, c, d) によって行われる．

この関数は，顧客エージェントのリストと，矩形範囲を表す 4 つの座標値を受け取り，範囲内に存在するアクティブなエージェント数を数える．ここで「アクティブ」とは，まだ入場前の個体や退場済みの個体を含めず，現在シミュレーション空間内で行動している個体のみを対象とするための条件である．具体的には person.is __ active を確認し，非アクティブ個体は集計から除外している．さらに座標引数の順序による影響を避けるために min/max を用いて x __ min, x __ max, y __ min, y __ max を決定しており，ドラッグ方向が左右・上下どちらであっても同じ結果となるように設計されている．

次に，人口密度の「分母」となる面積は，シミュレーション座標系における矩形の面積 sim __ area として算出される．ただし本プログラムでは，シミュレーション座標と実空間（m）との間にスケール差があるため，コメントに示されている通り「座標の 2 × 2 が実寸 1m × 1m」に相当するように補正している．すなわち，長さ方向の変換係数は 0.5，面積方向では $0.5 \times 0.5 = 0.25$ となるため，real __ area = sim __ area * 0.25 として実面積（m^2）へ変換している．

以上により，人口密度は「範囲内人数／実面積 (m^2)」として算出され，混雑度合いを物理的意味のある単位で評価できる．

人口密度の推移を表示するためのグラフ用ウィンドウは，必要時のみ作成する設計となっている．具体的には，グラフ描画の要素である fig __ density, ax __ density, line __ density を初期状態で None に設定し，範囲選択操作が行われた際に初めてウィンドウ生成を行う．

これにより，人口密度機能を利用しない場合には余計なウィンドウ生成や描画負荷を避けることができ，シミュレーション全体の操作性・軽量性の向上につながる．人口密度の測定対象範囲は，Matplotlib の RectangleSelector を用いてユーザが指定する．具体的には，メイン表示の Axes 上

で左クリックドラッグを行うと赤い半透明の矩形が表示され，ドラッグ完了時にコールバック関数 on __ select __ region(eclick, erelease) が呼び出される．

この関数内では，クリック開始点と終了点の座標（xdata, ydata）を取り出して (x1, x2, y1, y2) として保存し，これを selected __ region として保持する．したがって，以後のフレーム更新処理ではこの selected __ region が「人口密度を測定すべき領域」として参照される．さらに on __ select __ region では，人口密度推移を表示するための別ウィンドウ管理も同時に行っている．まず，人口密度グラフ用 Figure が未作成である，または一度閉じられて存在しない場合には plt.subplots(figsize=(6,4)) により新しい Figure を生成し，ウィンドウタイトルを「選択範囲の人口密度推移」に設定したうえで，折れ線グラフ用の line __ density を空データで作成し，fig __ density.show() によりウィンドウを表示する．一方，既にウィンドウが存在する場合は ax __ density.cla() により Axes をクリアし，同様に折れ線オブジェクトを再生成することで，以前の選択範囲の表示が残らないようにしている．

また，範囲を選び直したときに過去の密度推移を引き継いでしまうと比較が困難になるため，density __ history.clear() によって履歴データをリセットし，選択範囲に対応した新しい推移として記録をやり直す設計となっている．加えて，グラフにはタイトル（範囲の座標表示），横軸ラベル（時間），縦軸ラベル（人口密度 人/m^2）を設定し，fig __ density.canvas.draw() により表示を即座に更新している．人口密度の計算とグラフ更新は，アニメーション更新関数 update() 内で実行される．ただし，毎フレーム計算すると処理負荷が増大するため，UPDATE __ FREQUENCY = 44 を設け，「一定フレーム毎」にのみ計算する構造になっている．さらに，人口密度グラフのウィンドウが実際に開いている場合にのみ更新するように条件分岐が設けられており，fig __ density is not None かつ plt.fignum __ exists(fig __ density.number) を満たすときに限って計算を実行する．これにより，利用者が人口密度の可視化を必要としている場合のみ計算コストを払う設計となり，全体の効率が高められている．条件が成立すると，population __ density を呼び出して選択範囲の密度 d を算出し，それを density __ history に追加する．

その後，横軸データを履歴長に合わせて生成し，line __ density.set __ data(x __ data, density __ history) により折れ線のデータを更新する．また，密度の値は時刻により大きく変動する可能性があるため，ax __ density.relim() と ax __ density.autoscale __ view() により表示範囲を自動調整し，fig __ density.canvas.draw __ idle() により描画を反映する．この一連の処理により，利用者は選択範囲内の混雑度が時間とともに増減する様子を，折れ線グラフとして直感的に把握できる．加えて，本プログラムにはテスト用データを選択した場合の特別処理として，人口密度グラフを自動的に表示する機能が実装されている．具体的には，selected __ region を固定値 (24.0, 50.0, 0.0, 6.0) に設定し，on __ select __ region(None, None) を呼び出すことで，ドラッグ操作を行わなくても密度グラフウィンドウが生成されるようにしている．さらに，rect __ selector.extents に同じ範囲を設定することで，メイン画面側にも「どの範囲を測定しているか」が赤枠として視覚的に反映される．これにより，動作確認の段階でも人口密度推移の機能を確実に検証でき，分析機能を含む全体の動作テストが容易になる．

以上より，メイン処理内の人口密度機能は，選択範囲をユーザ操作で決定する（RectangleSelector），選択完了時に密度推移ウィンドウを生成・初期化する（on __ select __ region），update 処理内で一定間隔で人口密度を計算し履歴として蓄積する（population __ density + density __ history），折れ線グラフとして時間推移を表示するという流れで構成されている．これにより，レイアウト差や動線差が「混雑」という形でどの程度現れるかを，視覚的かつ定量的に比較できる分析基盤が提供されている．

（※文責: 菊地皓太）

A.4.5.6　衝突回数

```python
# パラメータ
COLLISION_SHOW_SECONDS = 1.0        # 衝突マークを表示し続ける時間（秒）
COLLISION_COOLDOWN_SECONDS = 2.0    # 一度衝突したペアの再カウントを防ぐ時間（秒）

# 状態管理変数
collision_markers = []              # 現在表示中のマーカーリスト
collision_history = {}              # ペアごとの最終衝突時刻
total_collision_count = 0           # 衝突総回数

# フレーム換算（アニメーション間隔と仮想時間倍率を反映）
collision_life_frames = int(COLLISION_SHOW_SECONDS * 1000 / ANIMATION_INTERVAL)
collision_cooldown_frames = int(COLLISION_COOLDOWN_SECONDS * 1000 / ANIMATION_INTERVAL)

# 描画用オブジェクト（赤い^^e2^^9c^^96印）
collision_scatter = ax.scatter([], [], color='red', marker='x', s=200, linewidth=4, zorder=20, label='Collision')

# カウンター表示用テキスト
collision_text = ax.text(0.12, 0.98, f"Collisions: 0", transform=ax.transAxes,
                         fontsize=10, va='top', ha='left', color='red',
                         bbox=dict(facecolor='white', alpha=0.6, edgecolor='red'), zorder=100)

rax = fig.add_axes([0.83, 0.4, 0.15, 0.4])
labels = ['テーブル', '椅子', 'オブジェクト', '斥力点',
          'セットルート', 'カレールート', 'ラーメンルート',
          '椅子用中継点', '向かう椅子', '衝突位置・回数']
visibility = [True, True, True, False, False, False, False, False, True, True]
check = CheckButtons(rax, labels, visibility)

def update_visibility(label):
    """チェックボックスの状態に応じてオブジェクトの表示/非表示を切り替える"""
    ...
```

```
    # 修正: '衝突位置・回数' がクリックされたら即座に反映させる
    if label == '衝突位置・回数':
        try:
            idx = labels.index('衝突位置・回数')
            is_visible = check.get_status()[idx]
            collision_text.set_visible(is_visible)
            collision_scatter.set_visible(is_visible)
            fig.canvas.draw_idle()
        except ValueError:
            pass
        return

check.on_clicked(update_visibility)

# アクティブかつ着席していないエージェントを抽出
active_agents = [c for c in customers if c.is_active and not c.reached_goal
↪  and not c.is_seated]

# ペアごとに距離をチェック
for c1, c2 in combinations(active_agents, 2):
    # 無敵時間チェック
    immunity_time = c1.params['collision_immunity_seconds']
    if c1.time_since_active < immunity_time or c2.time_since_active <
    ↪  immunity_time:
        continue

    dist = np.linalg.norm(c1.pos - c2.pos)

    # 衝突判定
    if dist < (AGENT_RADIUS):
        pair_id = tuple(sorted((c1.id, c2.id)))

        # クールダウン期間中でなければカウント
        last_frame = collision_history.get(pair_id, -9999)
        if frame - last_frame > collision_cooldown_frames:
            total_collision_count += 1
            collision_history[pair_id] = frame

            midpoint = (c1.pos + c2.pos) / 2
            collision_markers.append({
                'pos': midpoint,
                'life': collision_life_frames
```

```
            })

# マーカーの寿命を減らし、消えたものを除去
active_markers_pos = []
next_markers = []
for marker in collision_markers:
    marker['life'] -= 1
    if marker['life'] > 0:
        active_markers_pos.append(marker['pos'])
        next_markers.append(marker)
collision_markers[:] = next_markers # リスト更新

# チェックボックスの状態を取得して表示を制御
show_collision_info = check.get_status()[9] # リストの最後に追加した項目

# カウンターテキストの更新
collision_text.set_text(f"Collisions: {total_collision_count}")
collision_text.set_visible(show_collision_info)

# 衝突マークの更新
if active_markers_pos and show_collision_info:
    active_markers_np = np.array(active_markers_pos)
    collision_scatter.set_offsets(active_markers_np)
    collision_scatter.set_visible(True)
else:
    collision_scatter.set_visible(False)
```

本プログラムでは，食堂内の人流シミュレーションにおいて，エージェント同士が近接しすぎた状況を「衝突」とみなし，その発生回数を累積して表示する機能を実装している．この機能の目的は，単にエージェントの移動を可視化するだけでなく，混雑や接触リスクを定量的に評価できるようにすることである．すなわち，レイアウトや待ち行列方式（1 列／2 列）の違いによって，衝突がどの程度増減するかを比較する指標として利用できる．

衝突回数表示機能は，大きく分けて初期化（パラメータ・状態変数・描画要素の準備），UI による表示切替，update 関数内での衝突判定・カウント・表示更新，の三段階で構成されている．

まず初期化では，衝突の表示方法とカウント方法に関する時間パラメータが定義される．具体的には，衝突位置に描画する「×印」を何秒間表示し続けるかを COLLISION __ SHOW __ SECONDS で指定し，同一ペアの連続カウントを防止するためのクールダウン時間を COLLISION __ COOLDOWN __ SECONDS で指定している．このクールダウンの導入は重要であり，例えば 2 人がしばらく密着している状況で毎フレーム衝突を加算してしまうと，実態以上に衝突回数が過大評価される問題が生じる．そこで「同じペアの衝突は一定時間（一定フレーム数）を空けないと再度カウントしない」という条件を設け，指標としての妥当性を高めている．

次に状態管理変数として，現在表示中の衝突マーカー情報を保持する collision __ markers，ペア

ごとの最終衝突フレームを記録する collision __ history，衝突総回数を累積する total __ collision __ count が用意される．collision __ markers は「どこに×印を表示すべきか」「あと何フレーム表示を維持するか」という情報を持つリストであり，表示の寿命管理に使用される．一方 collision __ history は，「ペア ID の最後に衝突したフレーム番号」を対応付けて保持し，クールダウン判定に用いられる．

さらに，これらの時間指定を，実際のアニメーション更新周期に合わせたフレーム数へ変換する処理が行われる．ここで ANIMATION __ INTERVAL は「何ミリ秒ごとに update が呼ばれるか」を表すため，秒，ミリ秒，フレームの換算を行うことで「×印の寿命（フレーム数)」「クールダウン期間（フレーム数)」が決定される．これにより，衝突表示や衝突カウントの制御が，アニメーションの実行速度に依存しない形で一貫して動作する．

描画面では，衝突地点を示す赤い×印は collision __ scatter（scatter オブジェクト）として生成され，衝突回数は collision __ text（Axes 上のテキスト）として生成される．テキストは ax.transAxes 座標系で配置されているため，図の表示範囲（xlim/ylim）が変化しても，常に画面左上付近に固定されて表示される．これにより利用者は，シミュレーション中に視点を移動しても衝突回数を継続的に確認できる．

次に UI との連携として，本プログラムでは CheckButtons に「衝突位置・回数」という表示切替項目を追加し，衝突情報を必要に応じてオン／オフできるようにしている．特に update __ visibility 内では，該当ラベルがクリックされた場合に，チェック状態を参照して collision __ text および collision __ scatter の可視状態を即時に切り替え，fig.canvas.draw __ idle() によって描画反映を行う．これにより，衝突情報が不要な場合は非表示化して画面の視認性を確保し，必要な場合のみ表示して評価に利用する運用が可能となる．衝突の判定とカウントは，アニメーション更新関数 update() の内部で毎フレーム実行される．対象となるのは「現在行動中（is __ active）であり，退場済みではなく（not reached __ goal)，かつ着席中ではない（not is __ seated)」エージェントである．

このように対象を限定している理由は，着席中の個体は基本的に移動せず密集して見える可能性があるため，衝突判定を行うと混雑評価が歪む恐れがあること，また退場済み（ゴール到達）の個体を含める必要がないことにある．したがって，「移動や行列形成など衝突が問題となり得る場面」に評価対象を絞っていると解釈できる．

判定処理では，対象エージェント集合から 2 体ずつのペアを列挙し，互いの距離を計算する．その際入場直後の不自然な接触を抑制するために「無敵時間（immunity)」が導入されており，アクティブ化してからの経過時間 time __ since __ active が collision __ immunity __ seconds より短い場合は衝突判定をスキップする．これは，エージェントが入口付近に同時に生成される（あるいは近接して生成される）状況で，開始直後に衝突が多発してしまう不自然さを緩和する工夫である．衝突判定そのものは，2 者間距離 dist が閾値より小さいかどうかで行われる．本コードでは dist ¡ (AGENT __ RADIUS) を条件としているため，「互いの中心距離がエージェント半径未満になった場合」を衝突とみなしている．ここでの閾値設定はモデル化の一部であり，エージェントサイズや混雑判定の厳しさに直結するため，評価方針に応じて調整可能である．衝突が検出されると，まずペア ID（2 人の ID を昇順に並べたタプル）を生成し，collision __ history を用いてクールダウン期間を満たすかどうかを判定する．条件を満たす場合に限り total __ collision __ count を 1 増加させ，同時にそのペアの最終衝突フレームを更新する．これにより，同一ペアの過剰カウントが抑制される．さらに衝突位置表示のために，2 者の中点を計算し，life を付与したマーカー情報として collision __ markers に追加する．

したがって，衝突が起きた瞬間だけではなく，その後しばらく（指定フレーム数）衝突地点に×印が残る表示となり，利用者は「どこで衝突が起こりやすいか」を視覚的に把握できる．次に，毎フレームの終盤では collision __ markers に含まれる全マーカーの寿命を 1 ずつ減算し，寿命が尽きたマーカーを除外することで表示の自然な消滅を実現している．そして残存しているマーカー座標群を active __ markers __ pos として取り出し，表示 ON の場合に限って collision __ scatter.set __ offsets(...) により散布図の点群座標を更新する．これにより，衝突表示は「最新の衝突地点の集合」としてフレームごとに更新される．一方，表示 OFF または表示すべきマーカーがない場合は collision __ scatter を不可視化し，画面上に衝突情報が残らないようにしている．また，衝突回数テキストについても同様に，毎フレーム collision __ text.set __ text(...) により累積回数を更新し，チェックボックス状態に応じて可視・不可視を切り替えている．この結果として，衝突回数はシミュレーションの進行と同期して逐次更新され，利用者は衝突の発生傾向をリアルタイムで確認できる．

以上より，本プログラムの衝突回数機能は，クールダウンと表示寿命を備えた衝突イベント管理，UI による表示制御，update 内でのペア距離判定による衝突検出と累積回数更新，衝突地点の可視化（×印）と回数表示（テキスト）を同一フレーム更新で同期させる構成によって実現されている．これにより，レイアウトや導線条件を変更した際に「混雑・接触リスク」がどの程度変化するかを，視覚的かつ定量的に比較できる評価指標として活用可能となっている．

（※文責: 菊地皓太）

A.4.6　レイアウトデータ

本シミュレーションでは，食堂内の環境（入口・出口・レジ・机・椅子・壁・通路など）を layout.py にまとめ，get __ layout(layout __ name) により選択されたレイアウトに応じた辞書データ（layout __ data）として返す設計になっている．共通要素（入口/出口等，壁線分，通路，ルート）をベースに，Original / Layout A / Layout B の違いは主に机（TABLES）・椅子座標（CHAIR __ POSITIONS）・座席へ向かう中継点（CHAIR __ WAYPOINTS）・座席関連ルート（各 PATTERNS）として与えられる．

（※文責: 菊地皓太）

A.4.6.1　共通コンポーネント

```
# オブジェクトの定義
COMMON_POSITIONS = {
    "REGISTER_POSITIONS": np.array([[32, 8], [32, 9]]),
    "EXIT_POS": np.array([50, 0]),
    "ENTRANCE_POS": np.array([40, 0]),
    "RETURN_BOX_POS": np.array([17, 17]),
    "WATER_SERVER_POS": np.array([23.5, 18]),
    "TRASH_BOX_POS": np.array([48.5, 0.5]),
    "TERRACE_EXIT_POS": np.array([69.75, 13]),
}
```

```python

# 線分の定義
COMMON_WALLS_LINE = [
    {"type": "line", "start": np.array([9, 3], dtype=float), "end": np.array([24, 3], dtype=float)},
    {"type": "line", "start": np.array([9, 6], dtype=float), "end": np.array([24, 6], dtype=float)},
    {"type": "line", "start": np.array([15, 11], dtype=float), "end": np.array([28, 11], dtype=float)},
    {"type": "line", "start": np.array([15, 14], dtype=float), "end": np.array([28, 14], dtype=float)},
    {"type": "line", "start": np.array([13, 16], dtype=float), "end": np.array([15, 14], dtype=float)},
    {"type": "line", "start": np.array([0, 30], dtype=float), "end": np.array([70, 30], dtype=float)},
    {"type": "line", "start": np.array([0, 0], dtype=float), "end": np.array([0, 30], dtype=float)},
    {"type": "line", "start": np.array([70, 0], dtype=float), "end": np.array([70, 30], dtype=float)},
    {"type": "line", "start": np.array([0, 0], dtype=float), "end": np.array([39, 0], dtype=float)},
    {"type": "line", "start": np.array([41, 0], dtype=float), "end": np.array([49, 0], dtype=float)},
    {"type": "line", "start": np.array([51, 0], dtype=float), "end": np.array([70, 0], dtype=float)},
]

# 矩形の定義
COMMON_SPACES_RECT = [
    {"type": "rect", "pos": np.array([0, 0], dtype=float), "width": 3, "height": 17},
    {"type": "rect", "pos": np.array([0, 17], dtype=float), "width": 23, "height": 13},
    {"type": "rect", "pos": np.array([45.5, 0], dtype=float), "width": 2, "height": 2},
    {"type": "rect", "pos": np.array([68, 0], dtype=float), "width": 2, "height": 2},
    {"type": "rect", "pos": np.array([12, 8], dtype=float), "width": 18, "height": 1},
    {"type": "rect", "pos": np.array([34, 7.75], dtype=float), "width": 2, "height": 1.5},
```

```
    {"type": "rect", "pos": np.array([3, 8], dtype=float), "width": 10,
    ↪  "height": 1}
]

# セットルート
WAYPOINTS_1 = [
    {"pos": np.array([30, 1]), "wait": [0, 0]},
    {"pos": np.array([3.5, 1]), "wait": [0.5, 1.0]},
    {"pos": np.array([3.5, 2.5]), "wait": [0.5, 1.0]},
    {"pos": np.array([9.0, 7.25]), "wait": [0, 0]},
    {"pos": np.array([32, 7]), "wait": [0.5, 1.0]},
    {"pos": np.array([35, 7]), "wait": [0.5, 1.0]},
    {"pos": np.array([38, 7.5]), "wait": [0, 0]},
    {"pos": np.array([29, 16]), "wait": [0, 0]},
    {"pos": np.array([17, 16]), "wait": [0.5, 1.0]},
    {"pos": np.array([33, 15]), "wait": [0, 0]},
    {"pos": np.array([47, 5]), "wait": [0, 0]},
]

# カレールート
WAYPOINTS_2 = [
    {"pos": np.array([30, 4.5]), "wait": [0, 0]},
    {"pos": np.array([3.5, 4]), "wait": [0.5, 1.0]},
    {"pos": np.array([3.5, 5.5]), "wait": [0.5, 1.0]},
    {"pos": np.array([9.0, 7.25]), "wait": [0, 0]},
    {"pos": np.array([32, 7]), "wait": [0.5, 1.0]},
    {"pos": np.array([35, 7]), "wait": [0.5, 1.0]},
    {"pos": np.array([38, 7.5]), "wait": [0, 0]},
    {"pos": np.array([29, 16]), "wait": [0, 0]},
    {"pos": np.array([17, 16]), "wait": [0.5, 1.0]},
    {"pos": np.array([33, 15]), "wait": [0, 0]},
    {"pos": np.array([47, 5]), "wait": [0, 0]},
]

# ラーメンルート
WAYPOINTS_3 = [
    {"pos": np.array([38, 12]), "wait": [0, 0]},
    {"pos": np.array([3.5, 13]), "wait": [0.5, 1.0]},
    {"pos": np.array([3.5, 11.5]), "wait": [0.5, 1.0]},
    {"pos": np.array([12, 10]), "wait": [0, 0]},
    {"pos": np.array([32, 10]), "wait": [0.5, 1.0]},
    {"pos": np.array([35, 10]), "wait": [0.5, 1.0]},
```

```
    {"pos": np.array([35.5, 10]), "wait": [0, 0]},
    {"pos": np.array([29, 16]), "wait": [0, 0]},
    {"pos": np.array([17, 16]), "wait": [0.5, 1.0]},
    {"pos": np.array([33, 15]), "wait": [0, 0]},
    {"pos": np.array([47, 5]), "wait": [0, 0]},
]
```

本シミュレーションでは，レイアウトの比較実験（Original / A / B）を行うために，レイアウトごとに異なる部分（テーブル配置・座席ルート等）と，すべてのレイアウトに共通する部分（固定設備・壁・立入不可領域など）を分離して管理している．このうち 共通コンポーネントとは，どのレイアウトを選択しても不変である要素をまとめたものであり，主に以下の 3 つから構成される．

・固定設備の座標（COMMON __ POSITIONS）

・壁（線分）の定義（COMMON __ WALLS __ LINE）

・立入不可領域（矩形）の定義（COMMON __ SPACES __ RECT）

このように，共通部分をまとめて定義しておくことで，レイアウトごとの差分を明確化でき，修正時の影響範囲を小さくできる（同じ値を複数箇所に書かずに済む）という利点がある．

固定設備は，入口・出口・レジ・返却口など，エージェントが向かう目的地や行動の節目となる重要地点である．これらの設備は「面積を持つ家具」とは異なり，シミュレーション上は座標 1 点（または複数点）で表現できるため，ここでは NumPy 配列として位置情報を保持している．

REGISTER __ POSITIONS：レジ

レジは複数台存在する想定のため，[[x1, y1], [x2, y2]] の形で (N,2) 配列となっている．これにより draw __ registers() 側で register __ positions __ data[:,0] のようにまとめて描画できる．

EXIT __ POS / ENTRANCE __ POS：出口 / 入口

出入口は 1 箇所ずつのため [x, y] の (2,) 配列で保持する．

RETURN __ BOX __ POS：食器返却口

食器返却の行動を行うための目的地であり，座席行動後のルートにも関わる．

WATER __ SERVER __ POS：ウォーターサーバー

給水行動を表現する場合に利用する中間目的地として機能する．

TRASH __ BOX __ POS：ゴミ箱

エージェントの行動には直接関係しない．

TERRACE __ EXIT __ POS：テラス出入口

外部領域（テラス）との行き来を表すための出入口位置である．

COMMON __ WALLS __ LINE：壁（線分）

壁は空間の境界や通路の制約を表し，エージェントの移動可能領域を決める重要要素である．ここでは壁を「線分」として扱い，各壁を「type: ”line”」「start: 始点座標」「end: 終点座標」の辞書形式で統一して定義している．

・外周壁の表現

上辺：[0,30] → [70,30]

左辺：[0,0] → [0,30]

右辺：[70,0] → [70,30]

これにより，エージェントがシミュレーション領域外へ出ないように境界を設定している．

・出入口の開口を表現する分割

下辺（y=0）については，壁が「[0,0] → [39,0]」「[41,0] → [49,0]」「[51,0] → [70,0]」のように分割されている．これは，入口・出口が存在する区間を「壁を置かない開口部」として表現するためである．すなわち，壁を連続させずに隙間を作ることで，出入口の通行可能領域を作っている．

壁の座標は dtype=float として保持している．これは，エージェントの位置更新や補間計算（距離計算や線分上の点生成など）が実数演算になるため，整数型のままだと計算の一貫性が崩れる可能性があることを避ける意図がある．

COMMON __ SPACES __ RECT：立入不可領域（矩形）

線分壁だけでは表しにくい「面積を持つ障害物」や「立入不可スペース」を表現するため，矩形領域を別途定義している．矩形は「type: ”rect”」「pos: 左下座標」「width: 幅」「height: 高さ」で統一して表す．

矩形領域は主に以下のような用途に相当する．

・厨房側など，人が侵入できない領域

・カウンター下や棚下など，通り抜けできない領域

・柱や設備など，点や線ではなく面として扱うべき障害物

共通コンポーネントを独立させることで，以下の利点が得られる．

・レイアウト差分が明確になる．

A / B の変更点が「テーブル・椅子・ルート」に集中し，比較しやすい．

・保守性が高い．

入口位置など共通要素の修正が 1 箇所で済む．

・実験の公平性を確保できる．

出入口・返却口の位置などの前提条件を揃えたまま，座席配置だけ変える等の比較が可能となる．

WAYPOINTS：各ルート中継点

本シミュレーションでは，エージェントが食堂内を移動する際に，単純に「入口→レジ→座席」のような直線移動をさせると，

・テーブルや壁を突っ切る

・狭い通路で障害物に引っかかる

・不自然な角度で曲がる（壁際に突っ込む）

・実際の人間の導線からかけ離れる

などの問題が発生しやすい．

そのため本実装では，エージェントが通るべき経路を複数の中継点（waypoint）列として定義し，エージェントはそれらを順に辿ることで自然な導線を形成する．

この中継点列が WAYPOINTS __ 1，WAYPOINTS __ 2，WAYPOINTS __ 3 であり，食堂内の受け取り口の種類（セット／カレー／ラーメン）に応じて異なるルートを選択する仕組みになっている．

WAYPOINTS は，以下の形式の辞書を要素とするリストである．

”pos”: np.array([x, y]), ”wait”: [a, b]

pos：位置

・pos は座標 (x, y) を表す．

・エージェントはこの座標を「次の目的地」として順に移動する．

wait：待機時間

・wait はその地点での待機時間を [最小値, 最大値] で表す．

・実行時にはこの範囲でランダムな待機時間を決定する．

waypoint は単なる通過点ではなく，「ここで一瞬止まる」「列に並ぶ」などの人間らしい行動を再現するための制御点となっている．

（※文責: 菊地皓太）

A.4.6.2 original レイアウト定義

```
# テーブルの定義
TABLES_ORIGINAL = [
    {"type": "rect", "pos": np.array([30, 28], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([42, 28], dtype=float), "width": 6, "height": 2},
    {"type": "rect", "pos": np.array([51, 28], dtype=float), "width": 15, "height": 2},
    {"type": "rect", "pos": np.array([68, 18], dtype=float), "width": 2, "height": 9},
    {"type": "rect", "pos": np.array([68, 4], dtype=float), "width": 2, "height": 6},
    {"type": "rect", "pos": np.array([52, 0], dtype=float), "width": 12, "height": 2},
    {"type": "rect", "pos": np.array([55, 23], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([44, 23], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([33, 23], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([55, 18], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([44, 18], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([36, 18], dtype=float), "width": 6, "height": 2},
    {"type": "rect", "pos": np.array([55, 4], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([50, 4], dtype=float), "width": 3, "height": 2},
    {"type": "rect", "pos": np.array([55, 8], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([44, 8], dtype=float), "width": 9, "height": 2}
```

```
]

# 座席の定義
CHAIR_POSITIONS_ORIGINAL = []
_CHAIR_CONFIG_1_ORIGINAL = [
    {'y': 27.75, 'x_ranges': [np.linspace(31, 38, 6), np.linspace(43, 47, 4),
    ↪ np.linspace(52, 65, 10)]},
    {'y': 2.25, 'x_ranges': [np.linspace(52.5, 63.5, 8)]},
    {'x': 67.75, 'y_ranges': [np.linspace(4.5, 9.5, 4), np.linspace(18.5,
    ↪ 26.5, 6)]}
]
for config in _CHAIR_CONFIG_1_ORIGINAL:
    if 'y' in config:
        for x_vals in config['x_ranges']:
            for x in x_vals: CHAIR_POSITIONS_ORIGINAL.append(np.array([x,
            ↪ config['y']]))
    elif 'x' in config:
        for y_vals in config['y_ranges']:
            for y in y_vals:
            ↪ CHAIR_POSITIONS_ORIGINAL.append(np.array([config['x'], y]))
_CHAIR_CONFIG_2_ORIGINAL = {
    (22.75, 25.25): [np.linspace(33.5, 41.5, 9), np.linspace(44.5, 52.5, 9),
    ↪ np.linspace(55.5, 63.5, 9)],
    (17.75, 20.25): [np.linspace(36.5, 41.5, 6), np.linspace(44.5, 52.5, 9),
    ↪ np.linspace(55.5, 63.5, 9)],
    (7.75, 10.25): [np.linspace(44.5, 52.5, 9), np.linspace(55.5, 63.5, 9)],
    (3.75, 6.25): [np.linspace(50.5, 52.5, 3), np.linspace(55.5, 63.5, 9)]
}
for ys, x_ranges in _CHAIR_CONFIG_2_ORIGINAL.items():
    for y in ys:
        for x_vals in x_ranges:
            for x in x_vals: CHAIR_POSITIONS_ORIGINAL.append(np.array([x,
            ↪ y]))
```

テーブル定義：矩形（rect）として複数定義され，座標（pos）と幅（width）・高さ（height）で占有領域を表現している．座席位置の定義：椅子座標は，固定値を列挙するのではなく，np.linspace を用いた等間隔生成を組み合わせて大量の座席点を作る方式で定義されている．さらに，座席の塊（テーブルの周辺座席など）を複数の y 帯として管理するために，辞書__ CHAIR __ CONFIG __ 2 __ ORIGINAL を使って追加生成している．

（※文責: 菊地皓太）

A.4.6.3 original 座席中継点と各ルート一覧

```python
# 座席用の中継点
CHAIR_WAYPOINTS_ORIGINAL = {
    'P1': {"pos": np.array([65, 26]), "wait": [0, 0]},
    'P2': {"pos": np.array([65, 21.5]), "wait": [0, 0]},
    'P3': {"pos": np.array([65, 17]), "wait": [0, 0]},
    'P4': {"pos": np.array([54, 26]), "wait": [0, 0]},
    'P5': {"pos": np.array([54, 21.5]), "wait": [0, 0]},
    'P6': {"pos": np.array([54, 17]), "wait": [0, 0]},
    'P7': {"pos": np.array([43, 26]), "wait": [0, 0]},
    'P8': {"pos": np.array([43, 21.5]), "wait": [0, 0]},
    'P9': {"pos": np.array([43, 17]), "wait": [0, 0]},
    'P10': {"pos": np.array([32, 26]), "wait": [0, 0]},
    'P11': {"pos": np.array([35, 21.5]), "wait": [0, 0]},
    'P12': {"pos": np.array([35, 17]), "wait": [0, 0]},
    'P13': {"pos": np.array([65, 11]), "wait": [0, 0]},
    'P14': {"pos": np.array([65, 7]), "wait": [0, 0]},
    'P15': {"pos": np.array([65, 3]), "wait": [0, 0]},
    'P16': {"pos": np.array([54, 11]), "wait": [0, 0]},
    'P17': {"pos": np.array([54, 7]), "wait": [0, 0]},
    'P18': {"pos": np.array([54, 3]), "wait": [0, 0]},
    'P19': {"pos": np.array([43, 11]), "wait": [0, 0]},
    'P20': {"pos": np.array([43, 7]), "wait": [0, 0]},
    'P21': {"pos": np.array([43, 3]), "wait": [0, 0]},
    'P22': {"pos": np.array([49, 7]), "wait": [0, 0]},
    'P23': {"pos": np.array([49, 3]), "wait": [0, 0]}
}

# 7番目の中継点から座席に向かうルート一覧
CHAIR_AREA_PATTERNS_ORIGINAL = [
    {'x_range': (52, 65), 'y_range': (27.75, 27.75), 'patterns': [['P3',
    ↪  'P2', 'P1'], ['P6', 'P5', 'P4'], ['P9', 'P8', 'P7', 'P4'], ['P6',
    ↪  'P5', 'P2', 'P1'], ['P9', 'P8', 'P5', 'P4'], ['P9', 'P8', 'P5', 'P2',
    ↪  'P1'], ['P12', 'P11', 'P8', 'P5', 'P2', 'P1'], ['P12', 'P11', 'P8',
    ↪  'P5', 'P4'], ['P12', 'P11', 'P8', 'P7']]},
    {'x_range': (43, 47), 'y_range': (27.75, 27.75), 'patterns': [['P9',
    ↪  'P8', 'P7'], ['P12', 'P11', 'P8', 'P7']]},
    {'x_range': (31, 38), 'y_range': (27.75, 27.75), 'patterns': [['P10']]},
    {'x_range': (67.75, 67.75), 'y_range': (18.5, 26.5), 'patterns':
    ↪  [['P3']]},
```

```
34      {'x_range': (67.75, 67.75), 'y_range': (4.5, 9.5), 'patterns': [['P19',
        ↪  'P13']]},
35      {'x_range': (52.5, 63.5), 'y_range': (2.25, 2.25), 'patterns': [['P19',
        ↪  'P16', 'P17', 'P18'], ['P20', 'P23']]},
36      {'x_range': (55.5, 63.5), 'y_range': (25.25, 25.25), 'patterns': [['P3',
        ↪  'P2', 'P1'], ['P6', 'P5', 'P4'], ['P9', 'P8', 'P7', 'P4'], ['P6',
        ↪  'P5', 'P2', 'P1'], ['P9', 'P8', 'P5', 'P4'], ['P9', 'P8', 'P5', 'P2',
        ↪  'P1'], ['P12', 'P11', 'P8', 'P5', 'P2', 'P1'], ['P12', 'P11', 'P8',
        ↪  'P5', 'P4'], ['P12', 'P11', 'P8', 'P7']]},
37      {'x_range': (44.5, 52.5), 'y_range': (25.25, 25.25), 'patterns': [['P6',
        ↪  'P5', 'P4'], ['P9', 'P8', 'P7'], ['P12', 'P11', 'P8', 'P7']]},
38      {'x_range': (33.5, 41.5), 'y_range': (25.25, 25.25), 'patterns': [['P9',
        ↪  'P8', 'P7'], ['P12', 'P11', 'P8', 'P7'], ['P10']]},
39      {'x_range': (55.5, 63.5), 'y_range': (22.75, 22.75), 'patterns': [['P3',
        ↪  'P2'], ['P6', 'P5'], ['P9', 'P8', 'P5'], ['P12', 'P11', 'P8',
        ↪  'P5']]},
40      {'x_range': (44.5, 52.5), 'y_range': (22.75, 22.75), 'patterns': [['P6',
        ↪  'P5'], ['P9', 'P8'], ['P12', 'P11', 'P8']]},
41      {'x_range': (33.5, 41.5), 'y_range': (22.75, 22.75), 'patterns': [['P9',
        ↪  'P8'], ['P12', 'P11']]},
42      {'x_range': (55.5, 63.5), 'y_range': (20.25, 20.25), 'patterns': [['P3',
        ↪  'P2'], ['P6', 'P5'], ['P9', 'P8', 'P5'], ['P12', 'P11', 'P8',
        ↪  'P5']]},
43      {'x_range': (44.5, 52.5), 'y_range': (20.25, 20.25), 'patterns': [['P6',
        ↪  'P5'], ['P9', 'P8'], ['P12', 'P11', 'P8']]},
44      {'x_range': (36.5, 41.5), 'y_range': (20.25, 20.25), 'patterns': [['P9',
        ↪  'P8'], ['P12', 'P11']]},
45      {'x_range': (55.5, 63.5), 'y_range': (10.25, 10.25), 'patterns': [['P19',
        ↪  'P16']]},
46      {'x_range': (44.5, 52.5), 'y_range': (10.25, 10.25), 'patterns':
        ↪  [['P19']]},
47      {'x_range': (55.5, 63.5), 'y_range': (7.75, 7.75), 'patterns': [['P13',
        ↪  'P14'], ['P19', 'P16', 'P17'], ['P20', 'P22', 'P17']]},
48      {'x_range': (44.5, 52.5), 'y_range': (7.75, 7.75), 'patterns': [['P19',
        ↪  'P16', 'P17'], ['P20']]},
49      {'x_range': (55.5, 63.5), 'y_range': (6.25, 6.25), 'patterns': [['P13',
        ↪  'P14'], ['P19', 'P16', 'P17'], ['P20', 'P22', 'P17']]},
50      {'x_range': (50.5, 52.5), 'y_range': (6.25, 6.25), 'patterns': [['P17',
        ↪  'P16'], ['P20', 'P22']]},
51      {'x_range': (55.5, 63.5), 'y_range': (3.75, 3.75), 'patterns': [['P13',
        ↪  'P14', 'P15'], ['P19', 'P16', 'P17', 'P18'], ['P20', 'P23', 'P18']]},
```

```
    {'x_range': (50.5, 52.5), 'y_range': (3.75, 3.75), 'patterns': [['P19',
    ↪  'P16', 'P17', 'P18'], ['P20', 'P23']]},
    ]

# 座席から 8 番目の中継点へ向かうルート一覧
RETURN_ROUTE_PATTERNS_ORIGINAL = [
    {'x_range': (52, 65), 'y_range': (27.75, 27.75), 'patterns': [['P7',
    ↪  'P10'], ['P7', 'P8', 'P9'], ['P7', 'P8', 'P11'], ['P4', 'P5', 'P6'],
    ↪  ['P4', 'P5', 'P8', 'P9'], ['P4', 'P5', 'P8', 'P11'], ['P1', 'P2',
    ↪  'P3'], ['P1', 'P2', 'P5', 'P6'], ['P1', 'P2', 'P5', 'P8', 'P9'],
    ↪  ['P1', 'P2', 'P5', 'P8', 'P11']]},
    {'x_range': (43, 47), 'y_range': (27.75, 27.75), 'patterns': [['P10'],
    ↪  ['P7', 'P8', 'P9'], ['P7', 'P8', 'P11']]},
    {'x_range': (31, 38), 'y_range': (27.75, 27.75), 'patterns': [['P10']]},
    {'x_range': (67.75, 67.75), 'y_range': (18.5, 26.5), 'patterns':
    ↪  [['P3']]},
    {'x_range': (67.75, 67.75), 'y_range': (4.5, 9.5), 'patterns':
    ↪  [['P13']]},
    {'x_range': (52.5, 63.5), 'y_range': (2.25, 2.25), 'patterns': [['P23',
    ↪  'P20']]},
    {'x_range': (55.5, 63.5), 'y_range': (25.25, 25.25), 'patterns': [['P7',
    ↪  'P10'], ['P7', 'P8', 'P9'], ['P7', 'P8', 'P11'], ['P4', 'P5', 'P6'],
    ↪  ['P4', 'P5', 'P8', 'P9'], ['P4', 'P5', 'P8', 'P11'], ['P1', 'P2',
    ↪  'P3'], ['P1', 'P2', 'P5', 'P6'], ['P1', 'P2', 'P5', 'P8', 'P9'],
    ↪  ['P1', 'P2', 'P5', 'P8', 'P11']]},
    {'x_range': (44.5, 52.5), 'y_range': (25.25, 25.25), 'patterns': [['P4',
    ↪  'P5', 'P6'], ['P7', 'P8', 'P9'], ['P4', 'P5', 'P8', 'P9'], ['P4',
    ↪  'P5', 'P8', 'P11'], ['P7', 'P8', 'P11'], ['P10']]},
    {'x_range': (33.5, 41.5), 'y_range': (25.25, 25.25), 'patterns':
    ↪  [['P10'], ['P7', 'P8', 'P9'], ['P7', 'P8', 'P11']]},
    {'x_range': (55.5, 63.5), 'y_range': (22.75, 22.75), 'patterns': [['P2',
    ↪  'P3'], ['P5', 'P6'], ['P5', 'P8', 'P9'], ['P5', 'P8', 'P11']]},
    {'x_range': (44.5, 52.5), 'y_range': (22.75, 22.75), 'patterns': [['P5',
    ↪  'P6'], ['P8', 'P9'], ['P8', 'P11']]},
    {'x_range': (33.5, 41.5), 'y_range': (22.75, 22.75), 'patterns': [['P8',
    ↪  'P9'], ['P11']]},
    {'x_range': (55.5, 63.5), 'y_range': (20.25, 20.25), 'patterns': [['P2',
    ↪  'P3'], ['P5', 'P6'], ['P5', 'P8', 'P9'], ['P5', 'P8', 'P11']]},
    {'x_range': (44.5, 52.5), 'y_range': (20.25, 20.25), 'patterns': [['P5',
    ↪  'P6'], ['P8', 'P9'], ['P8', 'P11']]},
```

```
    {'x_range': (36.5, 41.5), 'y_range': (20.25, 20.25), 'patterns': [['P8', 'P9'], ['P11']]},
    {'x_range': (55.5, 63.5), 'y_range': (10.25, 10.25), 'patterns': [['P16']]},
    {'x_range': (44.5, 52.5), 'y_range': (10.25, 10.25), 'patterns': [['P19']]},
    {'x_range': (55.5, 63.5), 'y_range': (7.75, 7.75), 'patterns': [['P14', 'P13'], ['P17', 'P16'], ['P17', 'P22', 'P20']]},
    {'x_range': (44.5, 52.5), 'y_range': (7.75, 7.75), 'patterns': [['P17', 'P16'], ['P20']]},
    {'x_range': (55.5, 63.5), 'y_range': (6.25, 6.25), 'patterns': [['P14', 'P13'], ['P17', 'P16'], ['P17', 'P22', 'P20']]},
    {'x_range': (50.5, 52.5), 'y_range': (6.25, 6.25), 'patterns': [['P17', 'P16'], ['P20']]},
    {'x_range': (55.5, 63.5), 'y_range': (3.75, 3.75), 'patterns': [['P15', 'P14', 'P13'], ['P18', 'P17', 'P16'], ['P18', 'P17', 'P22', 'P20'], ['P18', 'P23', 'P20']]},
    {'x_range': (50.5, 52.5), 'y_range': (3.75, 3.75), 'patterns': [['P18', 'P17', 'P16'], ['P23', 'P20']]}
    ]

# ウォーターサーバーから座席へ向かうルート一覧
WATER_SERVER_RETURN_PATTERNS_ORIGINAL = [
    {'x_range': (52, 65), 'y_range': (27.75, 27.75), 'patterns': [['P11', 'P8', 'P5', 'P2', 'P1'], ['P11', 'P8', 'P5', 'P4'], ['P11', 'P8', 'P7', 'P4'], ['P10', 'P7', 'P4']]},
    {'x_range': (43, 47), 'y_range': (27.75, 27.75), 'patterns': [['P11', 'P8', 'P7'], ['P10', 'P7']]},
    {'x_range': (31, 38), 'y_range': (27.75, 27.75), 'patterns': [['P10']]},
    {'x_range': (67.75, 67.75), 'y_range': (18.5, 26.5), 'patterns': [['P12', 'P9', 'P6', 'P3'], ['P11', 'P8', 'P5', 'P2']]},
    {'x_range': (67.75, 67.75), 'y_range': (4.5, 9.5), 'patterns': [['P19', 'P13']]},
    {'x_range': (55.5, 63.5), 'y_range': (25.25, 25.25), 'patterns': [['P11', 'P8', 'P5', 'P2', 'P1'], ['P11', 'P8', 'P5', 'P4'], ['P11', 'P8', 'P7', 'P4'], ['P10', 'P7', 'P4']]},
    {'x_range': (44.5, 52.5), 'y_range': (25.25, 25.25), 'patterns': [['P11', 'P8', 'P5', 'P4'], ['P11', 'P8', 'P7']]},
    {'x_range': (33.5, 41.5), 'y_range': (25.25, 25.25), 'patterns': [['P11', 'P8', 'P7'], ['P10']]},
```

```
    {'x_range': (55.5, 63.5), 'y_range': (22.75, 22.75), 'patterns': [['P11', 'P8', 'P5']]},
    {'x_range': (44.5, 52.5), 'y_range': (22.75, 22.75), 'patterns': [['P11', 'P8']]},
    {'x_range': (33.5, 41.5), 'y_range': (22.75, 22.75), 'patterns': [['P11']]},
    {'x_range': (55.5, 63.5), 'y_range': (20.25, 20.25), 'patterns': [['P11', 'P8', 'P5']]},
    {'x_range': (44.5, 52.5), 'y_range': (20.25, 20.25), 'patterns': [['P11', 'P8']]},
    {'x_range': (36.5, 41.5), 'y_range': (20.25, 20.25), 'patterns': [['P11']]},
    {'x_range': (52.5, 63.5), 'y_range': (2.25, 2.25), 'patterns': [['P19', 'P16', 'P17', 'P18'], ['P20', 'P23']]},
    {'x_range': (55.5, 63.5), 'y_range': (10.25, 10.25), 'patterns': [['P19', 'P16']]},
    {'x_range': (44.5, 52.5), 'y_range': (10.25, 10.25), 'patterns': [['P19']]},
    {'x_range': (55.5, 63.5), 'y_range': (7.75, 7.75), 'patterns': [['P13', 'P14'], ['P19', 'P16', 'P17'], ['P20', 'P22', 'P17']]},
    {'x_range': (44.5, 52.5), 'y_range': (7.75, 7.75), 'patterns': [['P19', 'P16', 'P17'], ['P20']]},
    {'x_range': (55.5, 63.5), 'y_range': (6.25, 6.25), 'patterns': [['P13', 'P14'], ['P19', 'P16', 'P17'], ['P20', 'P22', 'P17']]},
    {'x_range': (50.5, 52.5), 'y_range': (6.25, 6.25), 'patterns': [['P17', 'P16'], ['P20', 'P22']]},
    {'x_range': (55.5, 63.5), 'y_range': (3.75, 3.75), 'patterns': [['P13', 'P14', 'P15'], ['P19', 'P16', 'P17', 'P18'], ['P20', 'P23', 'P18']]},
    {'x_range': (50.5, 52.5), 'y_range': (3.75, 3.75), 'patterns': [['P19', 'P16', 'P17', 'P18'], ['P20', 'P23']]},
]
```

CHAIR __ AREA __ PATTERNS __ ORIGINAL：食事受け取り後から座席へ向かうルート
RETURN __ ROUTE __ PATTERNS __ ORIGINAL：座席から返却動線へ向かうルート
WATER __ SERVER __ RETURN __ PATTERNS __ ORIGINAL：ウォーターサーバーから座席へ向かうルート

食堂内の座席エリアは，テーブルが島状に並び，通路幅も一定ではないため，エージェントが「目的座席へ直線移動」すると，
・テーブルに衝突しやすい.
・狭い通路で不自然な角度の回避が発生する.

・人の流れが一本化して渋滞が集中する．
といった問題が起こりやすい．

そのため本実装では，座席の位置（どの座席帯に属するか）に応じて，中継点（P1, P2, …）の列をあらかじめ複数用意し，そこから適切なルートを選択して移動させる設計としている．これにより，現実の導線に近い「通路に沿った移動」を再現し，同じ座席帯でも複数経路を選べることで混雑分散も表現できる．

x __ range, y __ range は「対象となる座席の座標範囲（座席帯）」を表す．エージェントが選んだ座席座標 (x, y) がこの範囲に含まれる場合，その座席帯に対応する patterns が候補となる．patterns は中継点ラベル（P 番号）の配列であり，エージェントが順に辿る waypoint の並びを表す．例えば ['P9', 'P8', 'P7', 'P6'] であれば，P9 → P8 → P7 → P6 の順に移動する．また，ここに書かれている P 番号は「中継点定義（CHAIR __ WAYPOINTS __ ORIGINAL）」側で座標を持つ．このルート一覧は「座標」ではなく「ラベル列」で管理することで，座標を一元管理でき，ルートの表現も簡潔になる．

これらのルート一覧データは，座席を座標範囲で分類し，分類ごとに複数の中継点列（patterns）を用意する設計である．

この方式には以下のような利点がある．
・座席配置に応じて「通路に沿った現実的な経路」を与えられる．
・同じ座席帯でも経路を分散して渋滞集中を抑えられる．
・行き（座席へ），帰り（返却へ），給水後（座席へ復帰）を分離して行動の非対称性を表現できる．

（※文責: 菊地皓太）

A.4.6.4　レイアウト A 定義

```
# テーブルの定義
TABLES_A = [
    {"type": "rect", "pos": np.array([30, 27], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([42, 27], dtype=float), "width": 6, "height": 2},
    {"type": "rect", "pos": np.array([51, 27], dtype=float), "width": 15, "height": 2},
    {"type": "rect", "pos": np.array([67, 17], dtype=float), "width": 2, "height": 9},
    {"type": "rect", "pos": np.array([67, 4], dtype=float), "width": 2, "height": 6},
    {"type": "rect", "pos": np.array([52, 1], dtype=float), "width": 12, "height": 2},
    {"type": "rect", "pos": np.array([55, 22], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([44, 22], dtype=float), "width": 9, "height": 2},
```

```
    {"type": "rect", "pos": np.array([33, 22], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([55, 17], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([44, 17], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([36, 17], dtype=float), "width": 6, "height": 2},
    {"type": "rect", "pos": np.array([55, 5], dtype=float), "width": 9, "height": 2},
    {"type": "rect", "pos": np.array([55, 9], dtype=float), "width": 9, "height": 2},
]

# 座席の位置の定義
CHAIR_POSITIONS_A = []
CHAIR_CONFIG_1 = [
    {'y': 29.25, 'x_ranges': [np.linspace(31, 38, 6), np.linspace(43, 47, 4), np.linspace(52, 65, 10)]},
    {'y': 26.75, 'x_ranges': [np.linspace(31, 38, 6), np.linspace(43, 47, 4), np.linspace(52, 65, 10)]},
    {'y': 3.25, 'x_ranges': [np.linspace(53, 63, 8)]},
    {'y': 0.75, 'x_ranges': [np.linspace(53, 63, 8)]},
    {'x': 69.25, 'y_ranges': [np.linspace(5, 9, 4), np.linspace(18, 25, 6)]},
    {'x': 66.75, 'y_ranges': [np.linspace(5, 9, 4), np.linspace(18, 25, 6)]}
]
for config in CHAIR_CONFIG_1:
    if 'y' in config:
        for x_vals in config['x_ranges']:
            for x in x_vals: CHAIR_POSITIONS_A.append(np.array([x, config['y']]))
    elif 'x' in config:
        for y_vals in config['y_ranges']:
            for y in y_vals: CHAIR_POSITIONS_A.append(np.array([config['x'], y]))
CHAIR_CONFIG_2 = {
    (21.75, 24.25): [np.linspace(33.5, 41.5, 9), np.linspace(44.5, 52.5, 9), np.linspace(55.5, 63.5, 9)],
    (16.75, 19.25): [np.linspace(36.5, 41.5, 6), np.linspace(44.5, 52.5, 9), np.linspace(55.5, 63.5, 9)],
    (8.75, 11.25): [np.linspace(55.5, 63.5, 9)],
```

```
    (4.75, 7.25): [np.linspace(55.5, 63.5, 9)]
}
for ys, x_ranges in CHAIR_CONFIG_2.items():
    for y in ys:
        for x_vals in x_ranges:
            for x in x_vals: CHAIR_POSITIONS_A.append(np.array([x, y]))
```

（※文責: 菊地皓太）

A.4.6.5　A 座席中継点と各ルート一覧

```
# 座席用の中継点
CHAIR_WAYPOINTS_A = {
    'Q1': {"pos": np.array([69.5, 26.5]), "wait": [0, 0]},
    'Q2': {"pos": np.array([69.5, 16.5]), "wait": [0, 0]},
    'Q3': {"pos": np.array([66.5, 29.5]), "wait": [0, 0]},
    'Q4': {"pos": np.array([66.5, 26.5]), "wait": [0, 0]},
    'Q5': {"pos": np.array([65.5, 25.5]), "wait": [0, 0]},
    'Q6': {"pos": np.array([65.5, 20.5]), "wait": [0, 0]},
    'Q7': {"pos": np.array([65.5, 15.5]), "wait": [0, 0]},
    'Q8': {"pos": np.array([49.5, 29.5]), "wait": [0, 0]},
    'Q9': {"pos": np.array([49.5, 26.5]), "wait": [0, 0]},
    'Q10': {"pos": np.array([54, 25.5]), "wait": [0, 0]},
    'Q11': {"pos": np.array([54, 20.5]), "wait": [0, 0]},
    'Q12': {"pos": np.array([54, 15.5]), "wait": [0, 0]},
    'Q13': {"pos": np.array([40.5, 29.5]), "wait": [0, 0]},
    'Q14': {"pos": np.array([40.5, 26.5]), "wait": [0, 0]},
    'Q15': {"pos": np.array([43, 25.5]), "wait": [0, 0]},
    'Q16': {"pos": np.array([43, 20.5]), "wait": [0, 0]},
    'Q17': {"pos": np.array([43, 15.5]), "wait": [0, 0]},
    'Q18': {"pos": np.array([29, 29.5]), "wait": [0, 0]},
    'Q19': {"pos": np.array([29, 26.5]), "wait": [0, 0]},
    'Q20': {"pos": np.array([32, 25.5]), "wait": [0, 0]},
    'Q21': {"pos": np.array([35, 20.5]), "wait": [0, 0]},
    'Q22': {"pos": np.array([35, 15.5]), "wait": [0, 0]},
    'Q23': {"pos": np.array([69.5, 10.5]), "wait": [0, 0]},
    'Q24': {"pos": np.array([69.5, 3.5]), "wait": [0, 0]},
    'Q25': {"pos": np.array([65, 12]), "wait": [0, 0]},
    'Q26': {"pos": np.array([65, 8]), "wait": [0, 0]},
    'Q27': {"pos": np.array([65, 4]), "wait": [0, 0]},
    'Q28': {"pos": np.array([65, 0.5]), "wait": [0, 0]},
    'Q29': {"pos": np.array([54, 12]), "wait": [0, 0]},
```

```
    'Q30': {"pos": np.array([54, 8]), "wait": [0, 0]},
    'Q31': {"pos": np.array([54, 4]), "wait": [0, 0]},
    'Q32': {"pos": np.array([51.5, 3.5]), "wait": [0, 0]},
    'Q33': {"pos": np.array([51.5, 0.5]), "wait": [0, 0]}
}

# 7番目の中継点から座席に向かうルート一覧
CHAIR_AREA_PATTERNS_A = [
    {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['Q19', 'Q18'], ['Q20', 'Q14', 'Q13'], ['Q17', 'Q16', 'Q15', 'Q14', 'Q13']]},
    {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['Q19', 'Q18', 'Q13'], ['Q17', 'Q16', 'Q15', 'Q14', 'Q13'], ['Q20', 'Q14', 'Q13'], ['Q20', 'Q15', 'Q9', 'Q8']]},
    {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['Q19', 'Q18', 'Q13', 'Q8'], ['Q20', 'Q15', 'Q10', 'Q5', 'Q4', 'Q3'], ['Q20', 'Q15', 'Q9', 'Q8'], ['Q7', 'Q6', 'Q5', 'Q4', 'Q3'], ['Q12', 'Q11', 'Q10', 'Q9', 'Q8'], ['Q17', 'Q16', 'Q15', 'Q9', 'Q8'], ['Q17', 'Q16', 'Q11', 'Q10', 'Q9', 'Q8'], ['Q17', 'Q16', 'Q11', 'Q10', 'Q5', 'Q4', 'Q3'], ['Q22', 'Q21', 'Q16', 'Q15', 'Q9', 'Q8']]},
    {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['Q20'], ['Q17', 'Q16', 'Q15'], ['Q22', 'Q21', 'Q16', 'Q15']]},
    {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['Q20', 'Q15'], ['Q17', 'Q16', 'Q15'], ['Q22', 'Q21', 'Q16', 'Q15']]},
    {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['Q7', 'Q6', 'Q5'], ['Q12', 'Q11', 'Q10'], ['Q17', 'Q16', 'Q15', 'Q10'], ['Q17', 'Q16', 'Q11', 'Q10'], ['Q22', 'Q21', 'Q16', 'Q11', 'Q10'], ['Q22', 'Q21', 'Q16', 'Q15', 'Q10'], ['Q22', 'Q21', 'Q16', 'Q11', 'Q6', 'Q5']]},
    {'x_range': (53, 63), 'y_range': (3.25, 3.25), 'patterns': [['Q31'], ['Q25', 'Q26', 'Q27']]},
    {'x_range': (53, 63), 'y_range': (0.75, 0.75), 'patterns': [['Q32', 'Q33'], ['Q25', 'Q26', 'Q27', 'Q28'], ['Q32', 'Q27', 'Q28']]},
    {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['Q29', 'Q25', 'Q23']]},
    {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['Q29', 'Q7', 'Q2']]},
    {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['Q29', 'Q25']]},
    {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['Q29', 'Q7']]},
```

```
52      {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['Q7',
        ↪  'Q6', 'Q5'], ['Q12', 'Q11', 'Q10'], ['Q12', 'Q11', 'Q6', 'Q5'],
        ↪  ['Q17', 'Q16', 'Q15', 'Q10'], ['Q17', 'Q16', 'Q11', 'Q10'], ['Q22',
        ↪  'Q21', 'Q16', 'Q11', 'Q10'], ['Q22', 'Q21', 'Q16', 'Q15', 'Q10']]},
53      {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['Q12',
        ↪  'Q11', 'Q10'], ['Q17', 'Q16', 'Q15'], ['Q17', 'Q16', 'Q11', 'Q10'],
        ↪  ['Q22', 'Q21', 'Q16', 'Q15'], ['Q22', 'Q21', 'Q16', 'Q11', 'Q10']]},
54      {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns':
        ↪  [['Q20'], ['Q17', 'Q16', 'Q15'], ['Q22', 'Q21', 'Q16', 'Q15']]},
55      {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['Q7',
        ↪  'Q6'], ['Q12', 'Q11'], ['Q17', 'Q16', 'Q11'], ['Q22', 'Q21', 'Q16',
        ↪  'Q11']]},
56      {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['Q12',
        ↪  'Q11'], ['Q17', 'Q16'], ['Q22', 'Q21', 'Q16']]},
57      {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns': [['Q17',
        ↪  'Q16'], ['Q22', 'Q21']]},
58      {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['Q7',
        ↪  'Q6'], ['Q12', 'Q11'], ['Q17', 'Q16', 'Q11'], ['Q22', 'Q21', 'Q16',
        ↪  'Q11']]},
59      {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['Q12',
        ↪  'Q11'], ['Q17', 'Q16'], ['Q22', 'Q21', 'Q16']]},
60      {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns': [['Q17',
        ↪  'Q16'], ['Q22', 'Q21']]},
61
62      # {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [[],
        ↪  [], []]},
63      # {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [[],
        ↪  [], []]},
64      # {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns': [[],
        ↪  [], []]},
65
66      {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns':
        ↪  [['Q29']]},
67      {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['Q30'],
        ↪  ['Q29', 'Q25', 'Q26']]},
68      {'x_range': (55.5, 63.5), 'y_range': (7.25, 7.25), 'patterns': [['Q30'],
        ↪  ['Q29', 'Q25', 'Q26']]},
69      {'x_range': (55.5, 63.5), 'y_range': (4.75, 4.75), 'patterns': [['Q31'],
        ↪  ['Q29', 'Q25', 'Q26', 'Q27']]}
70      ]
71
```

```
# 座席から 8 番目の中継点へ向かうルート一覧
RETURN_ROUTE_PATTERNS_A = [
    {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['Q18', 'Q19'], ['Q13', 'Q14', 'Q20']]},
    {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['Q13', 'Q18', 'Q19'], ['Q13', 'Q14', 'Q20'], ['Q8', 'Q9', 'Q15', 'Q20']]},
    {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['Q8', 'Q13', 'Q18', 'Q19'], ['Q8', 'Q9', 'Q15', 'Q20'], ['Q3', 'Q4', 'Q5', 'Q10', 'Q15', 'Q20']]},
    {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['Q20']]},
    {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['Q15', 'Q20']]},
    {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['Q10', 'Q15', 'Q20']]},
    {'x_range': (53, 63), 'y_range': (3.25, 3.25), 'patterns': [['Q31'], ['Q27', 'Q26', 'Q25', 'Q29']]},
    {'x_range': (53, 63), 'y_range': (0.75, 0.75), 'patterns': [['Q33'], ['Q28', 'Q27', 'Q26', 'Q25', 'Q29']]},
    {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['Q23', 'Q25', 'Q29']]},
    {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['Q2', 'Q7']]},
    {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['Q25', 'Q29']]},
    {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['Q7']]},
    {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['Q5', 'Q6', 'Q7', 'Q12', 'Q17', 'Q22'], ['Q10', 'Q15', 'Q20'], ['Q10', 'Q11', 'Q12', 'Q17', 'Q22'], ['Q10', 'Q11', 'Q16', 'Q17', 'Q22'], ['Q10', 'Q11', 'Q16', 'Q21', 'Q22']]},
    {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['Q10', 'Q11', 'Q12'], ['Q15', 'Q16', 'Q17'], ['Q15', 'Q16', 'Q21'], ['Q15', 'Q20']]},
    {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns': [['Q20'], ['Q15', 'Q16', 'Q17'], ['Q15', 'Q16', 'Q21']]},
    {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['Q6', 'Q7', 'Q12', 'Q17', 'Q22'], ['Q11', 'Q12', 'Q17', 'Q22'], ['Q11', 'Q16', 'Q17', 'Q22'], ['Q11', 'Q16', 'Q21', 'Q22']]},
    {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['Q11', 'Q12', 'Q17', 'Q22'], ['Q16', 'Q17', 'Q22'], ['Q16', 'Q21', 'Q22']]},
    {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns': [['Q21']]},
```

```
92      {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['Q11',
        ↪  'Q16', 'Q21'], ['Q11', 'Q12', 'Q17', 'Q22']]},
93      {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['Q16',
        ↪  'Q21'], ['Q16', 'Q17', 'Q22']]},
94      {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns':
        ↪  [['Q21']]},
95      {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [['Q12',
        ↪  'Q17', 'Q22']]},
96      {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [['Q17',
        ↪  'Q22']]},
97      {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns':
        ↪  [['Q22']]},
98      {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns':
        ↪  [['Q29']]},
99      {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['Q30'],
        ↪  ['Q26', 'Q25', 'Q29']]},
100     {'x_range': (55.5, 63.5), 'y_range': (7.25, 7.25), 'patterns': [['Q30'],
        ↪  ['Q26', 'Q25', 'Q29']]},
101     {'x_range': (55.5, 63.5), 'y_range': (4.75, 4.75), 'patterns': [['Q31'],
        ↪  ['Q27', 'Q26', 'Q25', 'Q29']]}
102     ]
103
104 # ウォーターサーバーから座席へ向かうルート一覧
105 WATER_SERVER_RETURN_PATTERNS_A = [
106     {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['Q19',
        ↪  'Q18'], ['Q20', 'Q14', 'Q13']]},
107     {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['Q19',
        ↪  'Q18', 'Q13'], ['Q20', 'Q14', 'Q13'], ['Q20', 'Q15', 'Q9', 'Q8']]},
108     {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['Q19',
        ↪  'Q18', 'Q13', 'Q8'], ['Q20', 'Q15', 'Q10', 'Q5', 'Q4', 'Q3'], ['Q20',
        ↪  'Q15', 'Q9', 'Q8']]},
109     {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['Q20'],
        ↪  ['Q21', 'Q16', 'Q15']]},
110     {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['Q20',
        ↪  'Q15'], ['Q21', 'Q16', 'Q15']]},
111     {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['Q21',
        ↪  'Q16', 'Q11', 'Q10'], ['Q21', 'Q16', 'Q15', 'Q10'], ['Q21', 'Q16',
        ↪  'Q11', 'Q6', 'Q5']]},
112     {'x_range': (53, 63), 'y_range': (3.25, 3.25), 'patterns': [['Q31'],
        ↪  ['Q29', 'Q25', 'Q26', 'Q27']]},
```

```
    {'x_range': (53, 63), 'y_range': (0.75, 0.75), 'patterns': [['Q32',
    ↪  'Q33'], ['Q29', 'Q25', 'Q26', 'Q27', 'Q28'], ['Q32', 'Q27', 'Q28']]},
    {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['Q29',
    ↪  'Q25', 'Q23']]},
    {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['Q29',
    ↪  'Q7', 'Q2']]},
    {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['Q29',
    ↪  'Q25']]},
    {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['Q29',
    ↪  'Q7']]},
    {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['Q21',
    ↪  'Q16', 'Q11', 'Q10'], ['Q21', 'Q16', 'Q15', 'Q10'], ['Q20', 'Q15',
    ↪  'Q10']]},
    {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['Q22',
    ↪  'Q21', 'Q16', 'Q15'], ['Q21', 'Q16', 'Q11', 'Q10'], ['Q20', 'Q15']]},
    {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns':
    ↪  [['Q20'], ['Q21', 'Q16', 'Q15']]},
    {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['Q21',
    ↪  'Q16', 'Q11']]},
    {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['Q21',
    ↪  'Q16']]},
    {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns':
    ↪  [['Q21']]},
    {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['Q21',
    ↪  'Q16', 'Q11']]},
    {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['Q21',
    ↪  'Q16']]},
    {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns':
    ↪  [['Q21']]},
    {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [['Q22',
    ↪  'Q17', 'Q12']]},
    {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [['Q22',
    ↪  'Q17']]},
    {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns':
    ↪  [['Q22']]},
    {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns':
    ↪  [['Q29']]},
    {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['Q30'],
    ↪  ['Q29', 'Q25', 'Q26']]},
    {'x_range': (55.5, 63.5), 'y_range': (7.25, 7.25), 'patterns': [['Q30'],
    ↪  ['Q29', 'Q25', 'Q26']]},
```

```
    {'x_range': (55.5, 63.5), 'y_range': (4.75, 4.75), 'patterns': [['Q31'],
    ↪  ['Q29', 'Q25', 'Q26', 'Q27']]}
    ]
```

（※文責: 菊地皓太）

A.4.6.6　レイアウト B 定義

```
# テーブルの定義
TABLES_B = [
    {"type": "rect", "pos": np.array([30, 27], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([42, 27], dtype=float), "width": 6,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([51, 27], dtype=float), "width": 15,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([67, 17], dtype=float), "width": 2,
    ↪  "height": 9},
    {"type": "rect", "pos": np.array([67, 4], dtype=float), "width": 2,
    ↪  "height": 6},
    {"type": "rect", "pos": np.array([52, 4], dtype=float), "width": 12,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([55, 22], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([44, 22], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([33, 22], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([55, 17], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([44, 17], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([36, 17], dtype=float), "width": 6,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([44, 9], dtype=float), "width": 9,
    ↪  "height": 2},
    {"type": "rect", "pos": np.array([55, 9], dtype=float), "width": 9,
    ↪  "height": 2},
]

# 座席の位置の定義
```

```
CHAIR_POSITIONS_B = []
CHAIR_CONFIG_1 = [
    {'y': 29.25, 'x_ranges': [np.linspace(31, 38, 6), np.linspace(43, 47, 4), np.linspace(52, 65, 10)]},
    {'y': 26.75, 'x_ranges': [np.linspace(31, 38, 6), np.linspace(43, 47, 4), np.linspace(52, 65, 10)]},
    {'y': 6.25, 'x_ranges': [np.linspace(53, 63, 8)]},
    {'y': 3.75, 'x_ranges': [np.linspace(53, 63, 8)]},
    {'x': 69.25, 'y_ranges': [np.linspace(5, 9, 4), np.linspace(18, 25, 6)]},
    {'x': 66.75, 'y_ranges': [np.linspace(5, 9, 4), np.linspace(18, 25, 6)]}
]
for config in CHAIR_CONFIG_1:
    if 'y' in config:
        for x_vals in config['x_ranges']:
            for x in x_vals: CHAIR_POSITIONS_B.append(np.array([x, config['y']]))
    elif 'x' in config:
        for y_vals in config['y_ranges']:
            for y in y_vals: CHAIR_POSITIONS_B.append(np.array([config['x'], y]))
CHAIR_CONFIG_2 = {
    (21.75, 24.25): [np.linspace(33.5, 41.5, 9), np.linspace(44.5, 52.5, 9), np.linspace(55.5, 63.5, 9)],
    (16.75, 19.25): [np.linspace(36.5, 41.5, 6), np.linspace(44.5, 52.5, 9), np.linspace(55.5, 63.5, 9)],
    (8.75, 11.25): [np.linspace(55.5, 63.5, 9),np.linspace(44.5, 52.5, 9)]
}
for ys, x_ranges in CHAIR_CONFIG_2.items():
    for y in ys:
        for x_vals in x_ranges:
            for x in x_vals: CHAIR_POSITIONS_B.append(np.array([x, y]))
```

（※文責: 菊地皓太）

A.4.6.7　B 座席中継点と各ルート一覧

```
# 座席用の中継点
CHAIR_WAYPOINTS_B = {
    'R1': {"pos": np.array([69.5, 26.5]), "wait": [0, 0]},
    'R2': {"pos": np.array([69.5, 16.5]), "wait": [0, 0]},
    'R3': {"pos": np.array([66.5, 29.5]), "wait": [0, 0]},
```

```
    'R4': {"pos": np.array([66.5, 26.5]), "wait": [0, 0]},
    'R5': {"pos": np.array([65.5, 25.5]), "wait": [0, 0]},
    'R6': {"pos": np.array([65.5, 20.5]), "wait": [0, 0]},
    'R7': {"pos": np.array([65.5, 15.5]), "wait": [0, 0]},
    'R8': {"pos": np.array([49.5, 29.5]), "wait": [0, 0]},
    'R9': {"pos": np.array([49.5, 26.5]), "wait": [0, 0]},
    'R10': {"pos": np.array([54, 25.5]), "wait": [0, 0]},
    'R11': {"pos": np.array([54, 20.5]), "wait": [0, 0]},
    'R12': {"pos": np.array([54, 15.5]), "wait": [0, 0]},
    'R13': {"pos": np.array([40.5, 29.5]), "wait": [0, 0]},
    'R14': {"pos": np.array([40.5, 26.5]), "wait": [0, 0]},
    'R15': {"pos": np.array([43, 25.5]), "wait": [0, 0]},
    'R16': {"pos": np.array([43, 20.5]), "wait": [0, 0]},
    'R17': {"pos": np.array([43, 15.5]), "wait": [0, 0]},
    'R18': {"pos": np.array([29, 29.5]), "wait": [0, 0]},
    'R19': {"pos": np.array([29, 26.5]), "wait": [0, 0]},
    'R20': {"pos": np.array([32, 25.5]), "wait": [0, 0]},
    'R21': {"pos": np.array([35, 20.5]), "wait": [0, 0]},
    'R22': {"pos": np.array([35, 15.5]), "wait": [0, 0]},
    'R23': {"pos": np.array([69.5, 10.5]), "wait": [0, 0]},
    'R24': {"pos": np.array([69.5, 3.5]), "wait": [0, 0]},
    'R25': {"pos": np.array([65, 12]), "wait": [0, 0]},
    'R26': {"pos": np.array([65, 8]), "wait": [0, 0]},
    'R27': {"pos": np.array([65, 3]), "wait": [0, 0]},
    'R28': {"pos": np.array([54, 12]), "wait": [0, 0]},
    'R29': {"pos": np.array([54, 8]), "wait": [0, 0]},
    'R30': {"pos": np.array([51, 3]), "wait": [0, 0]},
    'R31': {"pos": np.array([43, 12]), "wait": [0, 0]},
    'R32': {"pos": np.array([43, 8]), "wait": [0, 0]},
}

# 7番目の中継点から座席に向かうルート一覧
CHAIR_AREA_PATTERNS_B = [
    {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['R19',
    ↪  'R18'], ['R20', 'R14', 'R13'], ['R17', 'R16', 'R15', 'R14', 'R13']]},
    {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['R19',
    ↪  'R18', 'R13'], ['R17', 'R16', 'R15', 'R14', 'R13'], ['R20', 'R14',
    ↪  'R13'], ['R20', 'R15', 'R9', 'R8']]},
```

```
41      {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['R19',
        ↪  'R18', 'R13', 'R8'], ['R20', 'R15', 'R10', 'R5', 'R4', 'R3'], ['R20',
        ↪  'R15', 'R9', 'R8'], ['R7', 'R6', 'R5', 'R4', 'R3'], ['R12', 'R11',
        ↪  'R10', 'R9', 'R8'], ['R17', 'R16', 'R15', 'R9', 'R8'], ['R17', 'R16',
        ↪  'R11', 'R10', 'R9', 'R8'], ['R17', 'R16', 'R11', 'R10', 'R5', 'R4',
        ↪  'R3'], ['R22', 'R21', 'R16', 'R15', 'R9', 'R8']]},
42      {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['R20'],
        ↪  ['R17', 'R16', 'R15'], ['R22', 'R21', 'R16', 'R15']]},
43      {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['R20',
        ↪  'R15'], ['R17', 'R16', 'R15'], ['R22', 'R21', 'R16', 'R15']]},
44      {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['R7',
        ↪  'R6', 'R5'], ['R12', 'R11', 'R10'], ['R17', 'R16', 'R15', 'R10'],
        ↪  ['R17', 'R16', 'R11', 'R10'], ['R22', 'R21', 'R16', 'R11', 'R10'],
        ↪  ['R22', 'R21', 'R16', 'R15', 'R10'], ['R22', 'R21', 'R16', 'R11',
        ↪  'R6', 'R5']]},
45      {'x_range': (53, 63), 'y_range': (6.25, 6.25), 'patterns': [['R32',
        ↪  'R29'], ['R31', 'R28', 'R29'], ['R31', 'R28', 'R25', 'R26']]},
46      {'x_range': (53, 63), 'y_range': (3.75, 3.75), 'patterns': [['R32',
        ↪  'R30'], ['R31', 'R28', 'R25', 'R26', 'R27']]},
47      {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['R31',
        ↪  'R28', 'R25', 'R23']]},
48      {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['R31',
        ↪  'R12', 'R7', 'R2']]},
49      {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['R31',
        ↪  'R28', 'R25', 'R26']]},
50      {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['R31',
        ↪  'R12', 'R7']]},
51      {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['R7',
        ↪  'R6', 'R5'], ['R12', 'R11', 'R10'], ['R12', 'R11', 'R6', 'R5'],
        ↪  ['R17', 'R16', 'R15', 'R10'], ['R17', 'R16', 'R11', 'R10'], ['R22',
        ↪  'R21', 'R16', 'R11', 'R10'], ['R22', 'R21', 'R16', 'R15', 'R10']]},
52      {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['R12',
        ↪  'R11', 'R10'], ['R17', 'R16', 'R15'], ['R17', 'R16', 'R11', 'R10'],
        ↪  ['R22', 'R21', 'R16', 'R15'], ['R22', 'R21', 'R16', 'R11', 'R10']]},
53      {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns':
        ↪  [['R20'], ['R17', 'R16', 'R15'], ['R22', 'R21', 'R16', 'R15']]},
54      {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['R7',
        ↪  'R6'], ['R12', 'R11'], ['R17', 'R16', 'R11'], ['R22', 'R21', 'R16',
        ↪  'R11']]},
55      {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['R12',
        ↪  'R11'], ['R17', 'R16'], ['R22', 'R21', 'R16']]},
```

```
    {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns': [['R17', 'R16'], ['R22', 'R21']]},
    {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['R7', 'R6'], ['R12', 'R11'], ['R17', 'R16', 'R11'], ['R22', 'R21', 'R16', 'R11']]},
    {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['R12', 'R11'], ['R17', 'R16'], ['R22', 'R21', 'R16']]},
    {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns': [['R17', 'R16'], ['R22', 'R21']]},

    # {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [[], [], []]},
    # {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [[], [], []]},
    # {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns': [[], [], []]},

    {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns': [['R31', 'R28'], ['R32', 'R29', 'R28']]},
    {'x_range': (44.5, 52.5), 'y_range': (11.25, 11.25), 'patterns': [['R31'], ['R32', 'R29', 'R28']]},
    {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['R31', 'R28', 'R29'], ['R32', 'R29']]},
    {'x_range': (44.5, 52.5), 'y_range': (8.75, 8.75), 'patterns': [['R32'], ['R31', 'R28', 'R29']]}
    ]


# 座席から 8 番目の中継点へ向かうルート一覧
RETURN_ROUTE_PATTERNS_B = [
    {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['R18', 'R19'], ['R13', 'R14', 'R20']]},
    {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['R13', 'R18', 'R19'], ['R13', 'R14', 'R20'], ['R8', 'R9', 'R15', 'R20']]},
    {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['R8', 'R13', 'R18', 'R19'], ['R8', 'R9', 'R15', 'R20'], ['R3', 'R4', 'R5', 'R10', 'R15', 'R20']]},
    {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['R20']]},
    {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['R15', 'R20']]},
```

```
79      {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['R10',
        ↪  'R15', 'R20']]},
80      {'x_range': (53, 63), 'y_range': (6.25, 6.25), 'patterns': [['R29',
        ↪  'R32'], ['R29', 'R28', 'R31'], ['R26', 'R25', 'R28', 'R31']]},
81      {'x_range': (53, 63), 'y_range': (3.75, 3.75), 'patterns': [['R30',
        ↪  'R32'], ['R27', 'R26', 'R25', 'R28', 'R31']]},
82      {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['R23',
        ↪  'R25', 'R28', 'R31']]},
83      {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['R2',
        ↪  'R7']]},
84      {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['R25',
        ↪  'R28', 'R31']]},
85      {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['R7']]},
86      {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['R5',
        ↪  'R6', 'R7', 'R12', 'R17', 'R22'], ['R10', 'R15', 'R20'], ['R10',
        ↪  'R11', 'R12', 'R17', 'R22'], ['R10', 'R11', 'R16', 'R17', 'R22'],
        ↪  ['R10', 'R11', 'R16', 'R21', 'R22']]},
87      {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['R10',
        ↪  'R11', 'R12'], ['R15', 'R16', 'R17'], ['R15', 'R16', 'R21'], ['R15',
        ↪  'R20']]},
88      {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns':
        ↪  [['R20'], ['R15', 'R16', 'R17'], ['R15', 'R16', 'R21']]},
89      {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['R6',
        ↪  'R7', 'R12', 'R17', 'R22'], ['R11', 'R12', 'R17', 'R22'], ['R11',
        ↪  'R16', 'R17', 'R22'], ['R11', 'R16', 'R21', 'R22']]},
90      {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['R11',
        ↪  'R12', 'R17', 'R22'], ['R16', 'R17', 'R22'], ['R16', 'R21', 'R22']]},
91      {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns':
        ↪  [['R21']]},
92      {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['R11',
        ↪  'R16', 'R21'], ['R11', 'R12', 'R17', 'R22']]},
93      {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['R16',
        ↪  'R21'], ['R16', 'R17', 'R22']]},
94      {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns':
        ↪  [['R21']]},
95      {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [['R12',
        ↪  'R17', 'R22']]},
96      {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [['R17',
        ↪  'R22']]},
97      {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns':
        ↪  [['R22']]},
```

```
98      {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns': [['R28',
        ↪  'R31']]},
99      {'x_range': (44.5, 52.5), 'y_range': (11.25, 11.25), 'patterns':
        ↪  [['R31']]},
100     {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['R29',
        ↪  'R32'], ['R29', 'R28', 'R31'], ['R26', 'R25', 'R28', 'R31']]},
101     {'x_range': (44.5, 52.5), 'y_range': (8.75, 8.75), 'patterns': [['R32'],
        ↪  ['R29', 'R28', 'R31']]}
102     ]
103
104
105 # ウォーターサーバーから座席へ向かうルート一覧
106 WATER_SERVER_RETURN_PATTERNS_B = [
107     {'x_range': (31, 38), 'y_range': (29.25, 29.25), 'patterns': [['R19',
        ↪  'R18'], ['R20', 'R14', 'R13']]},
108     {'x_range': (43, 47), 'y_range': (29.25, 29.25), 'patterns': [['R19',
        ↪  'R18', 'R13'], ['R20', 'R14', 'R13'], ['R20', 'R15', 'R9', 'R8']]},
109     {'x_range': (52, 65), 'y_range': (29.25, 29.25), 'patterns': [['R19',
        ↪  'R18', 'R13', 'R8'], ['R20', 'R15', 'R10', 'R5', 'R4', 'R3'], ['R20',
        ↪  'R15', 'R9', 'R8']]},
110     {'x_range': (31, 38), 'y_range': (26.75, 26.75), 'patterns': [['R20']]},
111     {'x_range': (43, 47), 'y_range': (26.75, 26.75), 'patterns': [['R20',
        ↪  'R15'], ['R21', 'R16', 'R15']]},
112     {'x_range': (52, 65), 'y_range': (26.75, 26.75), 'patterns': [['R20',
        ↪  'R15', 'R10'], ['R21', 'R16', 'R11', 'R10'], ['R21', 'R16', 'R15',
        ↪  'R10'], ['R21', 'R16', 'R11', 'R6', 'R5']]},
113     {'x_range': (53, 63), 'y_range': (6.25, 6.25), 'patterns': [['R32',
        ↪  'R29'], ['R31', 'R28', 'R29'], ['R31', 'R28', 'R25', 'R26']]},
114     {'x_range': (53, 63), 'y_range': (3.75, 3.75), 'patterns': [['R32',
        ↪  'R30'], ['R31', 'R28', 'R25', 'R26', 'R27']]},
115     {'x_range': (69.25, 69.25), 'y_range': (5, 9), 'patterns': [['R31',
        ↪  'R28', 'R25', 'R23']]},
116     {'x_range': (69.25, 69.25), 'y_range': (18, 25), 'patterns': [['R22',
        ↪  'R17', 'R12', 'R7', 'R2']]},
117     {'x_range': (66.75, 66.75), 'y_range': (5, 9), 'patterns': [['R31',
        ↪  'R28', 'R25', 'R26']]},
118     {'x_range': (66.75, 66.75), 'y_range': (18, 25), 'patterns': [['R22',
        ↪  'R17', 'R12', 'R7']]},
119     {'x_range': (55.5, 63.5), 'y_range': (24.25, 24.25), 'patterns': [['R20',
        ↪  'R15', 'R10'], ['R21', 'R16', 'R11', 'R10'], ['R21', 'R16', 'R15',
        ↪  'R10']]},
```

```
    {'x_range': (44.5, 52.5), 'y_range': (24.25, 24.25), 'patterns': [['R20', 'R15'], ['R21', 'R16', 'R15']]},
    {'x_range': (33.5, 41.5), 'y_range': (24.25, 24.25), 'patterns': [['R20'], ['R21', 'R16', 'R15']]},
    {'x_range': (55.5, 63.5), 'y_range': (21.75, 21.75), 'patterns': [['R21', 'R16', 'R11']]},
    {'x_range': (44.5, 52.5), 'y_range': (21.75, 21.75), 'patterns': [['R21', 'R16']]},
    {'x_range': (33.5, 41.5), 'y_range': (21.75, 21.75), 'patterns': [['R21']]},
    {'x_range': (55.5, 63.5), 'y_range': (19.25, 19.25), 'patterns': [['R21', 'R16', 'R11']]},
    {'x_range': (44.5, 52.5), 'y_range': (19.25, 19.25), 'patterns': [['R21', 'R16']]},
    {'x_range': (36.5, 41.5), 'y_range': (19.25, 19.25), 'patterns': [['R21']]},
    {'x_range': (55.5, 63.5), 'y_range': (16.75, 16.75), 'patterns': [['R22', 'R17', 'R12']]},
    {'x_range': (44.5, 52.5), 'y_range': (16.75, 16.75), 'patterns': [['R22', 'R17']]},
    {'x_range': (36.5, 41.5), 'y_range': (16.75, 16.75), 'patterns': [['R22']]},
    {'x_range': (55.5, 63.5), 'y_range': (11.25, 11.25), 'patterns': [['R31', 'R28']]},
    {'x_range': (44.5, 52.5), 'y_range': (11.25, 11.25), 'patterns': [['R31']]},
    {'x_range': (55.5, 63.5), 'y_range': (8.75, 8.75), 'patterns': [['R31', 'R28', 'R29'], ['R32', 'R29']]},
    {'x_range': (44.5, 52.5), 'y_range': (8.75, 8.75), 'patterns': [['R32'], ['R31', 'R28', 'R29']]}
    ]
```

（※文責: 菊地皓太）

A.4.6.8　メイン関数

```
def get_layout(layout_name):

    """
    指定されたレイアウト名のデータを辞書として返す
    """
```

```
    layout_data = {
        **COMMON_POSITIONS,
        "WALLS_LINE": COMMON_WALLS_LINE,
        "SPACES_RECT": COMMON_SPACES_RECT,
        "WAYPOINTS_1": WAYPOINTS_1,
        "WAYPOINTS_2": WAYPOINTS_2,
        "WAYPOINTS_3": WAYPOINTS_3,
    }

    if layout_name == 'Layout A':
        layout_data["TABLES"] = TABLES_A
        layout_data["CHAIR_POSITIONS"] = CHAIR_POSITIONS_A
        layout_data["CHAIR_WAYPOINTS"] = CHAIR_WAYPOINTS_A
        layout_data["CHAIR_AREA_PATTERNS"] = CHAIR_AREA_PATTERNS_A
        layout_data["RETURN_ROUTE_PATTERNS"] = RETURN_ROUTE_PATTERNS_A
        layout_data["WATER_SERVER_RETURN_PATTERNS"] =
        ↪ WATER_SERVER_RETURN_PATTERNS_A

    elif layout_name == 'Layout B':
        layout_data["TABLES"] = TABLES_B
        layout_data["CHAIR_POSITIONS"] = CHAIR_POSITIONS_B
        layout_data["CHAIR_WAYPOINTS"] = CHAIR_WAYPOINTS_B
        layout_data["CHAIR_AREA_PATTERNS"] = CHAIR_AREA_PATTERNS_B
        layout_data["RETURN_ROUTE_PATTERNS"] = RETURN_ROUTE_PATTERNS_B
        layout_data["WATER_SERVER_RETURN_PATTERNS"] =
        ↪ WATER_SERVER_RETURN_PATTERNS_B

    else: # 'Original'
        layout_data["TABLES"] = TABLES_ORIGINAL
        layout_data["CHAIR_POSITIONS"] = CHAIR_POSITIONS_ORIGINAL
        layout_data["CHAIR_WAYPOINTS"] = CHAIR_WAYPOINTS_ORIGINAL
        layout_data["CHAIR_AREA_PATTERNS"] = CHAIR_AREA_PATTERNS_ORIGINAL
        layout_data["RETURN_ROUTE_PATTERNS"] = RETURN_ROUTE_PATTERNS_ORIGINAL
        layout_data["WATER_SERVER_RETURN_PATTERNS"] =
        ↪ WATER_SERVER_RETURN_PATTERNS_ORIGINAL

    # 選択されたレイアウトに基づいて AVAILABLE_CHAIRS を初期化
    available = [np.copy(p) for p in layout_data["CHAIR_POSITIONS"]]
    random.shuffle(available)
    layout_data["AVAILABLE_CHAIRS"] = available
```

```
45      return layout_data
```

本シミュレーションでは，食堂内の配置を複数パターン（Original / Layout A / Layout B）で比較できるようにするため，レイアウトに関するすべてのデータを一括して取得する関数 get __ layout(layout __ name) を実装している．この関数は，引数として受け取ったレイアウト名に応じて，各種レイアウトデータ（設備座標，壁，障害物，テーブル，椅子，ルート情報など）を 辞書（dictionary）形式にまとめて返す役割を持つ．

get __ layout(layout __ name) の目的は次の通りである．

・レイアウト名（文字列）を変えるだけで，食堂内配置を別パターンに変更できる．

・本体側は layout __ data[”TABLES”] など同じキー名でアクセスするだけでよく，レイアウト別の if 文を本体側に書かずに済む．

・同じシミュレーション条件のまま，家具配置だけ変えて結果を比較できる．

（※文責: 菊地皓太）

A.4.7　CSV ファイル

本シミュレータでは，エージェントの入場に実データによる計測結果を反映させるため，実際に計測したデータを含む CSV ファイルを使用している．CSV 形式は，数値データや時系列データを簡潔に記述できるほか，Python からの読み込みや他のソフトウェア（Excel 等）による確認や編集が容易であるという利点を持つ．そのため，本シミュレーションにおける入出力データの管理手段として適している．

（※文責: 菊地皓太）

A.4.7.1　CSV ファイル利用の目的

CSV ファイルを用いる主な目的は以下の通りである．

・シミュレーション条件の外部化プログラム内に数値を直接書き込むのではなく，データをファイルとして分離することで，条件変更を容易にする．

・時系列データの管理時刻ごとの人数変化や発生イベントなどを表形式で保持できる．

・結果の再利用・分析シミュレーション結果を CSV として保存することで，後処理や統計解析，グラフ作成に利用できる．

・可視化と計算の分離データ生成と描画や解析を分離し，プログラム構造を整理できる．

（※文責: 菊地皓太）

A.4.7.2　CSV ファイルの形式とコード内での利用

本プロジェクトで用いる CSV ファイルは，行目にヘッダ，2 行目以降にデータを持つ表形式となっている．

time：時刻（秒，またはステップ数）

count：入場者数

このような構成により，「ある時刻に何人が存在しているか」「一定時間ごとに何人が入場・退場

したか」といった情報を時系列で管理できる．

CSV ファイルは主に Python の pandas や csv モジュールを用いて読み込ませ，読み込まれたデータは，DataFrame として保持され，エージェントの入場スケジュールの決定に利用できる．

（※文責: 菊地皓太）

A.5　メンバーコメント

1022177 西尾柊太

私たちのプロジェクトでは，大学食堂内の混雑を緩和すべく緩和策の検証と提案を目的として活動してきた．一年間を通して，単純に機関が短かったなと思った．実地計測は６月の１ヶ月のみ，他の計測・観察，利用者アンケート，従業員インタビューもすぐに終わり，他の時間はすべてコード開発だったと思う．私が貢献できたのは実地計測と利用者アンケート，従業員インタビューと発表会のスライド作成のみであったが，他のメンバーは計測やアンケートはもちろん，コード開発や各々自分のできることを最大限の努力でやり切っていた印象である．最初は６人のみでメンバーの頭数が少なく，このプロジェクトは完遂できるのかという不安もあったのは正直なところであるが，今となってはこの６人で良かったのではないかと思っている．

（※文責: 西尾柊太）

1023075 菊地皓太

この１年間，大学食堂の混雑緩和という身近な課題に向き合い，人流シミュレータを作成することで混雑緩和策の検証と提案を目指して活動してきた．振り返ると，当初は「混雑している」という感覚的な問題意識から出発したが，活動を進めるにつれて，混雑とは何か，どこに原因があるのか，どのように仮説を立てて検証すべきかといった問いが次々に生まれ，プロジェクトとしての意義が深まっていったと感じる．

食堂の混雑緩和を目的に掲げた当初，私の中には「混雑を緩和したい」という直感的な思いがある一方で，それを定量的に説明する視点や，改善策を根拠をもって示す枠組みは十分に整っていなかった．そのため，単に混雑状況を観察するだけでなく，実際に入場者・退場者数を計測し，時間帯ごとの傾向を把握したうえで，どこで滞留や待機が発生しているのかを分析する必要があった．計測活動では想像以上に現場ならではの難しさがあり，人数を数えるだけでも，測定方法の統一，担当の分担，記録ミスへの対策，データ整理といった工程が伴った．地道な作業の積み重ねではあったが，こうした実データがあったからこそ，混雑という現象を言語化し，数値として捉えることができたと思う．

人流シミュレータの作成を通しては，モデル化の難しさと面白さの両方を実感した．食堂には複数のオブジェクトが存在し，人の目的や行動パターンも一様ではない．さらに，滞留や待ち行列の形成，歩行時の衝突回避といった要素も加わるため，単純な移動モデルでは実際の様子を十分に再現できないことが分かった．人流モデルの考え方を学びながら，シミュレーション上でエージェントが自然に移動し，壁や障害物を回避しながら待ち行列を形成できるよう，試行錯誤を重ねた．最初は「動かすこと」を目標としていたが，次第に「どのように動かすか」「どの程度現実の動きに近いか」といった質の部分へ意識が移り，単なる実装作業にとどまらず，数理モデル・アルゴリズム・現場観察を結びつける統合的な活動へと発展させることができた．これはプロジェクト学習ならではの経験であり，授業で学んだ知識が実社会の課題と結びついていくことを実感した．

私はこの 1 年間，プロジェクトリーダーを務めた．リーダーとしては，技術面以上に「チームとして動くこと」の意義を強く感じた．プロジェクトは個々の能力だけで成立するものではなく，目的を共有し，方針を確認し，役割分担を明確にしながら，継続的に進捗を積み重ねていく必要がある．しかし実際には，メンバーそれぞれの予定や得意分野が異なり，モチベーションにも波があるため，議論が停滞したり，タスクが曖昧なまま時間だけが過ぎたりすることもあった．特に「何をすべきか分からない時間」が生じることは，プロジェクト全体の進行停滞につながるため，リーダーとしてその状況をどのように打開するかを考え続けた．

そのような場面で重要だったのは，課題を言語化し，議論の論点を整理したうえで段階的に検討を進め，次の行動へつなげることであった．感覚的な議論を続けるのではなく，「目的に対して今何をすべきか」「何が不足しているのか」「次に検証すべきことは何か」を具体化していくことで，方針が固まり，円滑に進行するようになった．自分だけで最適な答えを導くことは難しく，メンバー全員が意見を出しやすい環境を整え，情報共有を徹底することが大切だと学んだ．議事録や先生方，TA さんからのフィードバックがプロジェクトの進捗を大きく促したのも，「共有可能な形で残すこと」がプロジェクトの推進力になるからだと実感した．

このプロジェクトで最終的に成果物の提示まで完遂できたのは，間違いなくメンバー全員の力があったからである．計測活動を地道に続けてくれた人，議論を整理して方向性を示してくれた人，進捗確認や細部のサポートを担ってくれた人，実装を形にしてくれた人，資料作成の完成度を高めてくれた人，雰囲気を明るくして発言しやすい環境を作ってくれた人，それぞれの貢献が積み重なり，プロジェクトとして成立した．6 人という少人数のチームであり，誰か一人でも欠けていれば，この成果には至らなかったと感じる．

また，先生方や TA さんの存在も，この活動において大きな支えであった．私たち学生だけでは視野が狭くなりがちな部分を，先生方の助言によって客観的に見直すことができた．特に，実現可能性や目的の明確化，モデル化の方向性，仮説立案など重要な局面でのフィードバックは，プロジェクトの質を大きく引き上げてくれたと感じる．

最後に，プロジェクトを通じて関わってくれたメンバー，先生方，TA さんに心から感謝している．この経験を通して得た学びや反省を，今後の活動にも活かしていきたい．

（※文責: 菊地皓太）

1023100 坂井康大

私は本プロジェクトにおいて，基本的に明確な担当を割り当てられることなく，主に仲間のサポート役として活動した．本プロジェクトはメンバーが少なかったため，個別に担当を決めるのではなく，全員で協力して作り上げる方針が取られていたからである．その中で，私が唯一主導して行った作業は，活動全体のまとめであった．私は次々と連続して発生する作業よりも，結果を整理・統合するような，一つの区切りを持つ作業の方が得意であり，その特性を生かした形で貢献した．

私が本プロジェクトを通して学んだことは，大きく分けて三つある．

一つ目は，実地調査の重要性である．今回のプロジェクトでは，実際に食堂を訪れ，入退場者数の計測や，食堂従業員への聞き取り調査を行った．調査を行わなくとも，ある程度は結果を予測できる部分もあったが，実際に調査を行ったことで，調査結果が予測通りであった場合でも，それは単なる想定ではなく，動かしがたい事実として成果を裏付ける強い根拠になるということや，予測できなかった結果については，どれだけ考えていても導き出すことは難しく，実地調査を行ったからこそ得られた知見であると実感できた．

二つ目は，長期間をかけて成果を出す経験の価値である．一年間という時間をかけて一つの成果を形にする経験は，これまであまり得られなかったものであり，非常に貴重な学びとなった．この経験は，今後の四年次に行う卒業研究へとつながる有意義なものになったと感じている．

三つ目は，個人では成し得なかった成果があったという点である．私は，可能であれば一人で作業を完結させたいと考える傾向があり，人に頼ることを苦手としてきた．しかし，本プロジェクトで得られた一年間の成果は，私一人では決して実現できなかったものだと強く感じている．その具体例として挙げられるのが，コードのマージ作業である．今回のプロジェクトではリーダーがマージを担当してくれたが，なぜそれが問題なく機能しているのか，私には理解できなかった．自分のコード同士を統合するだけでも難しい中で，他者が作成したコードをまとめ上げる作業の難しさを実感し，適材適所の重要性とともに，深い感謝の念を抱いた．

一方で，本プロジェクトにおける反省点もある．それは，結果として指示待ちの姿勢になってしまったことである．これまで私は，できる限り一人で作業を進めることを選択してきたため，グループワークそのものを避けてきた．その結果，主体的に全体を主導するという発想に至らず，仲間が主導して進めている作業の補助や，最終段階でのまとめを中心に関わる形となった．前期の段階から何らかの形で主導的な役割を担えていれば，結果的に効率が下がる可能性もあったかもしれないが，それも含めて良い経験になったのではないかと感じている．この点を，今後に生かすべき反省として受け止めたい．

（※文責: 坂井康大）

1023104 櫻井孔士

私は 2025 年プロジェクト学習を通して貢献したと思うことは，数理モデルの理解だと感じる。今まで使ったことのない式や文字の量に最初は壁を感じた．しかし，コードに写していくうちに式の理解が深まった．また，式をそのまま映すだけでなく，計算量が莫大にならないような式にすることにも注力することができた．単に式を使うことだけがモデル化することではないと学ぶこともできた．

今回のプロジェクトを通して，コミュニケーションの重要性を改めて実感しました．自分の知識や工夫だけでは解決が難しい課題に直面したときは，研究室のメンバーや教員に相談し，助けを求めることも大切だと考えます．また，計画性をもって研究を進める必要性も感じました．今回は計画や目標を十分に立てずに進めたため，途中で目的を見失う場面がありました．今後はこうした事態を防ぐため，研究内容の「目的」「重要性」「合理性」を明確にしたうえで進めていきたいです．

（※文責: 櫻井孔士）

1023118 島津諒雅

本プロジェクトでは，大学食堂の混雑状況をシミュレーションで可視化し，緩和策を提案することを目的として活動した．テーマ選定から実行まで全て学生主体で行うプロジェクトは私にとって初めての経験であり，当初は進行方法に戸惑う場面も多かった．特に，現状の課題分析や「何を『混雑』と定義するか」という前提条件の策定には苦労した．しかし，全プロジェクト中最少の 6 人であったが逆に，密に話し合いを重ねることで，全員の認識を深く擦り合わせながら前進することができた．シミュレーション構築においては，現実の現象を再現する難しさと面白さの両方を学んだ．実際に自ら食堂を利用して学生の心理的・物理的な行動パターンを検証し，それをプログラム上のエージェントの動きに反映させる作業は困難であったが，意図通りに動いた時の達成感は大きかった．

成果発表を終え，プロジェクトの終了に寂しさを感じるほど，充実した時間を過ごすことができた．Python を用いたシミュレーションの実装力だけでなく，現実の現象をモデル化する抽象化能力や，チームで一つの課題を解決しようとする協調性が養われたと感じた．この 1 年間で得た経験とチームワークの重要性は，私にとってかけがえのないものになったと感じる．この経験を糧に，今後の研究活動においても，課題に対して主体的に取り組んでいきたい．

最後に，本プロジェクトを進めるにあたり，未熟な私たちに指導・助言してくださった先生方および TA さんに深く感謝している．そして何より，1 年間互いに支え合ったプロジェクトメンバーの皆に心から感謝したい．この 6 人であったからこそプロジェクトを完遂できたと思う．

（※文責: 島津諒雅）

1023128 鈴木創太

1 年間，プロジェクト学習に向き合ってみて多くの成長と変化があった．食堂の利用方法について考え直してみたり，活動における参加姿勢の改善もあった．以前までは何気なく使っていた食堂だがプロジェクトから見つけた問題点や従業員の大変なことに少しでも手助けできるような行動を心がけるようになった．実際に計測やアンケート，インタビューの後に食堂に訪れると，実際に見たり聞いたりした問題点が目に見えるようになり，それを対策する行動をするようになった．また，プロジェクトとしての参加も初めの方に比べれば大きく貢献できたと感じてる．初めの方はプロジェクト全体が初対面だったため，どこかよそよそしかったり，お互いに参加が消極的で誰かがやるのを待っている人も多かったが，後半の方は積極的な参加や助け合いの姿勢が多く見えたと思う．メンバー 6 人と少数ながら協力して完遂できたと思う．

（※文責: 鈴木創太）