S4 Simulation System

Version 6.2

チュートリアル

目 次

1 はじめに **7**

2 銀行の窓口モデル **7**

2.1 銀行の窓口モデル 7
2.2 起動 8
2.3 プロジェクトの作成 9
2.4 モデルを開く 9
2.5 「フローアイテム」の配置 9
2.6 「生成」部品の配置 11
2.7 「ファシリティ」の配置 11
2.8 他の部品の配置 11
2.9 リンクの作成 14
2.10 「来店」の編集 14
2.11 「窓口利用」の編集 16
2.12 パラメータの編集 17
2.13 モデルの実行 18
2.14 サマリの表示 18
2.15 データの表示 19
2.16 ヒストグラムの表示 19
2.17 時系列プロットの表示 21
2.18 リアルタイムグラフ 21
2.19 記録部品の配置 23
2.20 記録部品の編集 24
2.21 窓口の混雑具合の比較 26
2.22 システムの挙動変化の分析 29

3 銀行の窓口モデル応用 **30**

3.1 モデル 30
3.2 部品の配置 30
3.3 リンクの作成 31
3.4 来店の編集 32
3.5 窓口の編集 34
3.6 窓口利用の編集 34
3.7 シミュレーション終了時間の設定 36
3.8 アニメーション機能 37
3.9 アニメーションパラメータ編集 37
3.10 アニメーションの実行 38

4 製造工程モデル 38
4.1 製造工程モデル . . . 38
4.2 プロジェクトの作成 . . . 39
4.3 フローアイテムの配置 . . . 40
4.4 生成部品の配置 . . . 40
4.5 ファシリティの配置 . . . 40
4.6 ファシリティ利用部品の配置 . . . 40
4.7 終端部品の配置 . . . 40
4.8 リンクの作成 . . . 41
4.9 パーツ供給の編集 . . . 41
4.10 装置の編集 . . . 42
4.11 工程の編集 . . . 42
4.12 出荷の編集 . . . 45
4.13 実行パラメータの編集 . . . 45
4.14 目的関数 . . . 47
4.15 モデルの実行 . . . 47

5 最適化 47
5.1 問題設定 . . . 48
5.2 パラメータ設定 . . . 48
5.3 部品の設定 . . . 48
5.4 目的関数 . . . 50
5.5 最適化設定 . . . 50
5.6 最適化実行 . . . 54

6 Lotka-Volterra 方程式 56
6.1 Lotka-Volterra モデル . . . 56
6.2 プロジェクトの作成 . . . 56
6.3 フローアイテムの配置 . . . 56
6.4 生成部品の配置 . . . 56
6.5 連続変数の配置 . . . 57
6.6 連続部品の配置 . . . 57
6.7 終端部品の配置 . . . 57
6.8 リアルタイムグラフ部品の配置 . . . 57
6.9 パラメータの設定 . . . 57
6.10 リンクの作成 . . . 59
6.11 生成部品の編集 . . . 59
6.12 Lotka-Volterra の編集 . . . 59
6.13 連続変数の編集 . . . 59
6.14 リアルタイムグラフ部品の編集 . . . 60

7 同期エージェント 63
7.1 ライフゲーム . . . 63
7.2 エージェントモデリング . . . 63
7.3 プロジェクトの作成 . . . 64
7.4 環境の作成 . . . 64
7.5 同期エージェントの作成 . . . 64
7.6 エージェントの環境の設定 . . . 67
7.7 エージェントの初期化処理の編集 . . . 67
7.8 エージェントのステップ処理の編集 . . . 67
7.9 エージェント集合の初期化処理の編集 . . . 70
7.10 エージェント集合の可視化の編集 . . . 71
7.11 モデルの実行 . . . 71

8 非同期エージェント 72
8.1 ツイッターの伝播シミュレーション . . . 74
8.2 エージェントモデリング . . . 74
8.3 プロジェクトの作成 . . . 75
8.4 環境の作成 . . . 75
8.5 非同期エージェントの作成 . . . 75
8.6 エージェントの環境の設定 . . . 76
8.7 エージェントの初期化処理の編集 . . . 76
8.8 エージェントのプロセス処理の編集 . . . 77
8.9 エージェント集合の初期化処理の編集 . . . 79
8.10 エージェント集合の可視化の編集 . . . 80
8.11 モデルパラメータの変更 . . . 81
8.12 モデルの実行 . . . 81
8.13 リアルタイムグラフ . . . 82
8.14 Gephi による配置 . . . 84
8.15 GraphML フォーマットグラフの配置 . . . 86
8.16 モデルの再実行 . . . 86

9 ソーシャルフォースモデル 86
9.1 概要 . . . 86
9.2 プロジェクトの作成 . . . 86
9.3 SFM エージェントの作成 . . . 88
9.4 SFM 環境の作成 . . . 88
9.5 SFM 環境の設定 . . . 88
9.6 環境の初期化後の処理 . . . 89
9.7 エージェントの環境の設定 . . . 90
9.8 エージェント集合の初期化処理 . . . 91

9.9 環境上のエージェント描画処理 92
9.10 パラメータの編集 92
9.11 モデルの実行 92

10 施設内人流モデル **93**
10.1 概要 93
10.2 プロジェクトの作成 93
10.3 環境部品の作成 94
10.4 SFM 地図の配置 94
10.5 SFM 地図エディタの起動 94
10.6 レイアウトの作成 95
10.7 経路地点の作成 98
10.8 エッジの作成 98
10.9 属性の設定 99
10.10SFM エージェントの配置 100
10.11SFM エージェントの設定 102
10.12シミュレーションの実行 103
10.133D アニメーションの表示 103
10.13.1 3D アニメーションの配置 103
10.13.2 3D アニメーションの設定 103
10.13.3 シミュレーションの実行 105
10.13.4 3D アニメーションの実行 105
10.13.5 3D アニメーションの視点切り替え 107
10.13.6 3D アニメーションの視点操作 108
10.13.7 3D アニメーションの再生操作 109
10.13.8 3D アニメーションの環境設定 109
10.14可視化ダッシュボードの作成 109
10.14.1 可視化用データの出力設定 110
10.14.2 ダッシュボードの起動 111
10.14.3 プロットコンポーネント 111
10.14.4 テキストコンポーネント 112
10.14.5 グリッド操作 112
10.14.6 ダッシュボードの保存・読み込み 112
10.14.7 ダッシュボードの終了 112

11 映画館人流モデル **113**
11.1 概要 113
11.2 プロジェクトの作成 113
11.3 SFM 地図の配置 113
11.4 SFM 地図エディタの起動 114

11.5 地図の初期化 114
11.6 背景画像の取り込み 114
11.7 領域の作成 115
11.8 属性の設定 116
11.9 経路地点の作成 117
11.10経路計算手法の設定 118
11.11SFM エージェントの配置 118
11.12SFM エージェントの設定 121
11.13指定領域内の人数を数える 126

12 状態空間モデル 130
12.1 概要 130
12.1.1 状態空間モデルについて 130
12.1.2 本チュートリアルで作成するプロジェクトについて . . 133
12.2 状態空間モデルの作成手順 134
12.3 プロジェクトの作成 134
12.4 環境の作成 135
12.5 入力データの用意 135
12.6 エージェントの作成 136
12.7 エージェントの環境・入力の設定 136
12.8 粒子の生成処理の設定 137
12.9 粒子の状態遷移の設定 139
12.10粒子の尤度計算の設定 140
12.11可視化処理の設定 140
12.12パラメータの設定 144
12.13モデルの実行 144
12.14モデルパラメータの最適化 149
12.14.1 パラメータの設定 150
12.14.2 粒子フィルタエージェントの設定 151
12.14.3 目的関数の設定 152
12.14.4 最適化設定 153
12.14.5 最適化実行 156

13 窓口モデル用部品 156
13.1 駅の窓口 157
13.2 テンプレートモデルのインポート 158
13.3 「駅の窓口」モデルを開く 158
13.4 「お客」部品 161
13.5 「来店」部品 161
13.6 「購入待ち」部品 162

13.7 「窓口」部品 162
13.8 「切符購入」部品 165
13.9 「退店」部品 165
13.10シミュレーションパラメータ 167
13.11出力結果 167
13.12スーパーのレジ 170
13.13テンプレートプロジェクトのインポート 171
13.14スーパーのレジモデルを開く 171
13.15「お客」部品 173
13.16「レジ到着」部品 173
13.17「レジ選択」部品 173
13.18「レジスター」部品 176
13.19「レジ」部品 176
13.20「退店」部品 178
13.21出力結果 178

14 OpenStreetMap を用いたネットワークシミュレーション 180
14.1 プロジェクトの準備 180
14.2 OpenStreetMap からネットワークデータを読み込む 183
14.2.1 データの準備 183
14.2.2 環境部品から読み込み 184
14.2.3 テスト実行 184
14.3 エージェント出現位置、移動方法の指定 185
14.3.1 エージェントの出現位置を指定する 185
14.3.2 エージェントの目的地を設定する。 186
14.3.3 いくつかの経路地点を順番に移動させる 188
14.3.4 いくつかの地点をランダムに回遊させる 190
参考文献 195

1 はじめに

S4 Simulation System を使えば、工場などの生産システム、サプライチェーンなどの流通システム、銀行の窓口や ATM、通信システム、交通システム、コールセンターなど、何かしらの待ちや混雑の発生する、確率的な振る舞いをするシステムを対象とした、シミュレーションを行う事が出来ます。

S4 Simulation System はシミュレーションモデルを効率的にモデル化出来るように、部品・資源・フローアイテム・リンクという汎用オブジェクトを用意しています。これらを配置・接続・編集する事で、モデルを表現します。

本チュートリアルでは、S4 Simulation System によるモデル作成方法と、簡単な分析方法をご紹介します。本チュートリアルは S4 Simulation System の全ての機能を網羅的に紹介するものではありませんが、本チュートリアルを読めば、S4 Simulation System の基本概念、操作方法を理解出来るはずです。

2 銀行の窓口モデル

まずはじめに、「銀行の窓口モデル」を作成し、分析を行ってみます。

2.1 銀行の窓口モデル

銀行の窓口モデルは銀行に来るお客をシミュレーションするモデルです。(図 1)

- お客がランダムに到着する。
- 銀行にはひとつの窓口があり、お客はその窓口でサービスを受ける。
- 窓口が空いていない場合、お客は待ち行列を作ってサービスを待つ。
- 先に並んだ人から順に窓口でサービスを受ける事が出来る。
- サービス時間はランダムである。

銀行の窓口はひとつしかないので、お客の待ち行列が出来ます。その待ち行列がどのぐらいになるのか、お客の待ち時間はどのぐらいになるのか、などをシミュレーションします。

実際にシミュレーションを行うには、もう少し厳密な設定が必要です。ここでは、

- お客の到着間隔は平均 60 秒の指数分布に従う。
- 窓口のサービス時間は平均 30 秒の指数分布に従う。

とします。

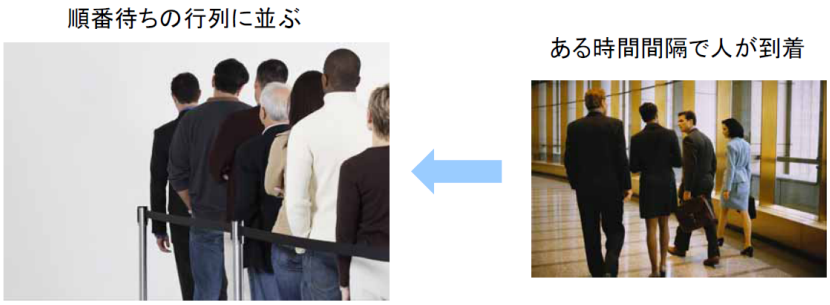


図 1: 銀行の窓口モデル

2.2　起動

まず、S4 Simulation System を起動します。S4 Simulation System をインストールすると、Windows のスタートメニューに登録されます。

Windows 7 以前の OS の場合、スタートメニューから、[すべてのプログラム]→[S-Quattro Simulation System]→[S-Quattro Simulation System] で起動します。(ただし、インストール時の設定で別のメニューに登録されている場合もあります。)

Windows 8, 8.1 の場合、スタート画面からアプリ画面を開き、そこから [S4(S-Quattro Simulation System)] で起動します。

Windows 10 以降の場合、スタートメニューから、[すべてのアプリ]→[MSI Solutions]→[S4(S-Quattro Simulation System)] で起動します。

サーバーにてご利用の場合、上記のいずれかの OS の方法で起動できます。

※なお、どの OS でも検索機能で「s4」と入力すれば概ね S4 Simulation System がヒットします。

初回起動時は、初回起動画面 (図 2) が表示されます。

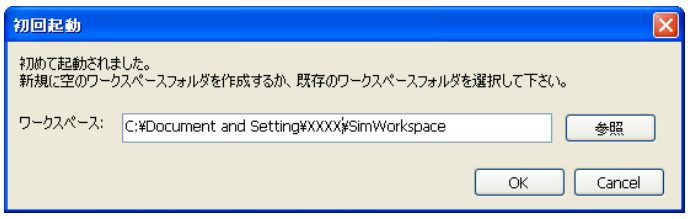


図 2: 初回起動

初回起動画面ではワークスペースフォルダを指定して下さい。ワークスペースフォルダとは、S4 Simulation System で作成したプロジェクトを保存するフォルダの事です。ワークスペースフォルダは後で変更する事も出来ます。よく分からなければ、デフォルトの設定のままで構いません。

2.3　プロジェクトの作成

ファイルメニューから「新規プロジェクト」を選択し、プロジェクトを作成します。(図 3)

図 3: プロジェクトの作成

[新規プロジェクトの作成」ダイアログ (図 4) が表示されますので、プロジェクト名「銀行の窓口」を入力して下さい。

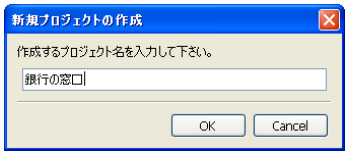


図 4: 新規プロジェクトダイアログ

2.4　モデルを開く

ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。(図 5)

モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。

2.5　「フローアイテム」の配置

ブラウザパネル (左側のパネル) で「アイテム」タブを選択して下さい。(図 6)

図 5: 作成されたプロジェクト

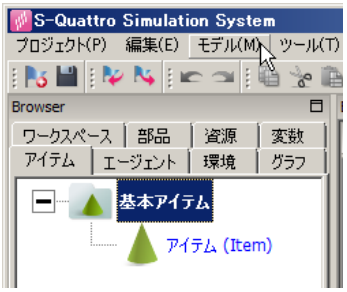


図 6: アイテムタブ

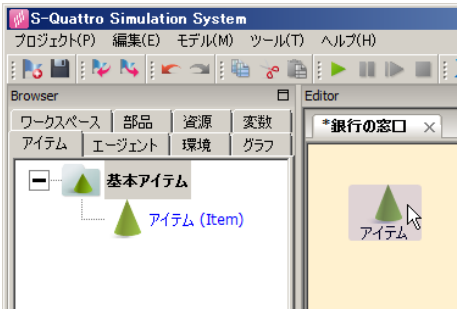


図 7: フローアイテムの配置

アイテムタブの「アイテム」上でマウスの左ボタンを押し、押しながら、モデル編集画面 (右側) に移動し、左ボタンを離すと配置されます。(図 7)

フローアイテムとはシミュレーション内に流れるオブジェクトで、この場合は、お客を示します。配置したフローアイテムの下側をダブルクリックするとフローアイテムの名前を変更する事が出来ます。ここでは「お客」という名前に変えます。(図 8)

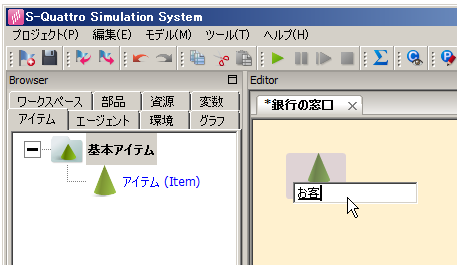


図 8: フローアイテムの名前の変更

2.6　「生成」部品の配置

次に来店を表す「生成」部品を配置します。ブラウザパネル (左側のパネル) で「部品タブ」を選択して下さい。(図 9)

フローアイテムの配置した方法と同様に、「生成」部品をモデル上に配置します。この部品は来店を表すので、フローアイテムの名前を変更した方法と同様に「来店」という名前に変更します。(図 10)

2.7　「ファシリティ」の配置

次に窓口を表す資源「ファシリティ」部品を配置します。ファシリティは、排他的に利用可能な複数の離散資源を表します。例えば銀行の窓口がそれにあたりますが、他にはオフィス内のプリンタ、生産ライン上の同時にひとつしか処理出来ない工程などもファシリティです。

ブラウザパネル (左側のパネル) で「資源」タブを選択して下さい。(図 11)

資源タブからファシリティをモデル上に配置します。この部品は窓口を表すので、「窓口」という名前に変更します。(図 12)

2.8　他の部品の配置

同様に部品タブから「ファシリティ利用」部品を「窓口利用」という名前で配置し、「終端」部品を「退店」という名前で配置します。(図 13)

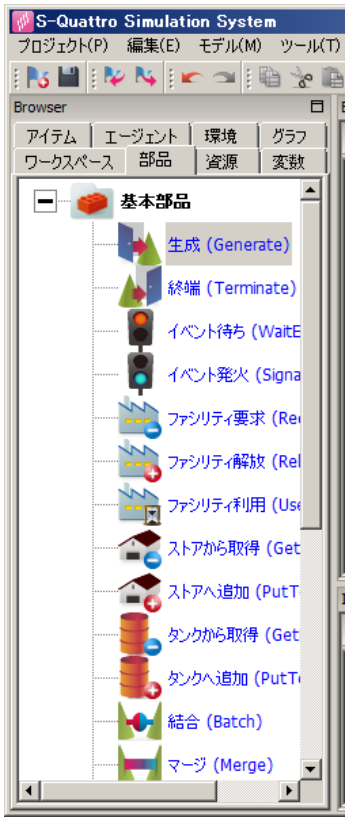


図 9: 部品タブ

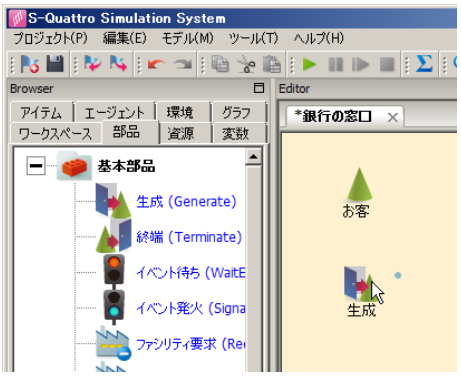


図 10: 生成部品の配置

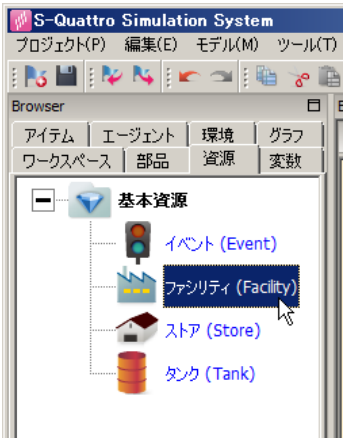


図 11: 資源タブ

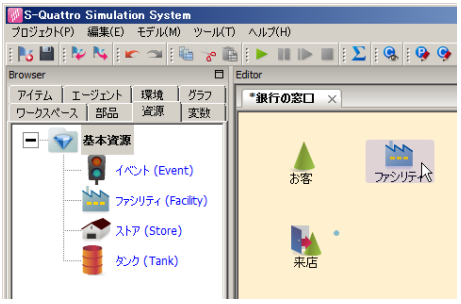


図 12: ファシリティの配置

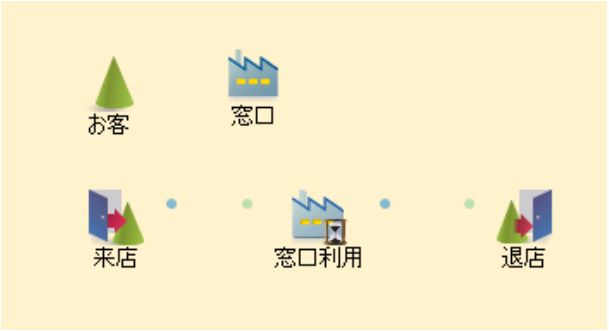


図 13: 他の部品の配置

2.9　リンクの作成

次にお客の流れをリンクで表現します。「来店」部品の右側の丸で、マウスの左ボタンを押し、押しながら、「窓口利用」部品の左側の丸に移動し、左ボタンを離すと、リンクが作成されます。(図 14)

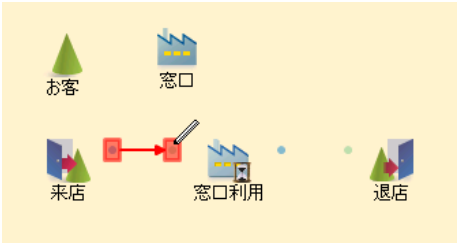


図 14: リンクの作成

同様に、「窓口利用」部品の右側の丸から「退店」部品に左側の丸にもリンクを作成します。(図 15)

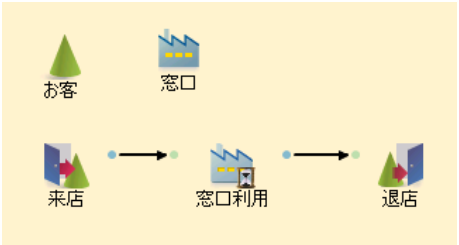


図 15: リンクの作成 (2)

2.10　「来店」の編集

次に「来店」部品の振る舞いを指定します。「来店」部品をダブルクリックすると部品編集画面が表示されます。(図 16)

「来店」部品ではある間隔で「お客」が到着する事を表現します。言い換えれば「お客」をある間隔で生成します。「お客」を生成するためには「フローアイテム名」のプルダウンメニューから「お客」を選択します。(図 17)

お客の到着間隔は平均 60 秒の指数分布に従う事になっていました。そこで、「生成時間」の「生成方式」を「指数分布」にし、「平均」を 60 に設定します (図 18)。

注意: 生成部品の「初期生成時間」は、生成を開始する時間を指定します。ここでは、生成する間隔をランダムに設定したいので、「生成時間」の方を指定します。

図 16: 「来店」の編集画面

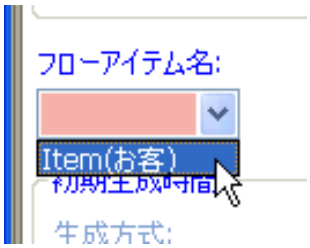


図 17: フローアイテムの選択

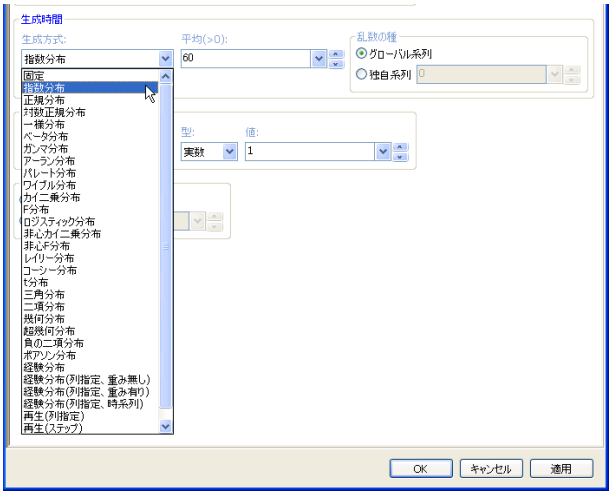


図 18: 分布の選択

他の項目はそのままにし、「OK」を押すと、部品編集画面が閉じます。

2.11　「窓口利用」の編集

次に「窓口利用」部品の振る舞いを指定します。「窓口利用」部品をダブルクリックすると部品編集画面が表示されます。(図 19)

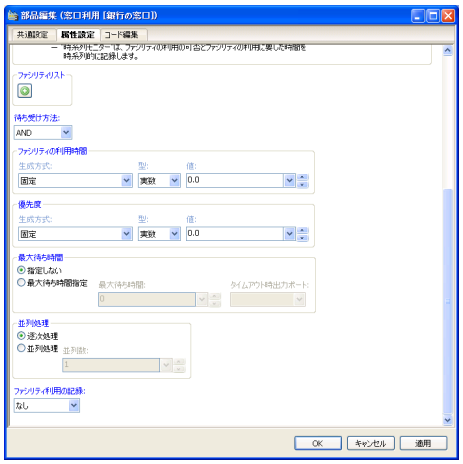


図 19:「窓口利用」の編集

「窓口利用」はファシリティ「窓口」をある時間占有する事を示します。「ファシリティリスト」の「+」をクリックし、プルダウンメニューから「窓口」を選択します。

窓口のサービス時間は平均 30 秒の指数分布に従う事になっていました。そこで、「ファシリティの利用時間」の「生成方式」を指数分布にし、「平均」を 30 に設定します。

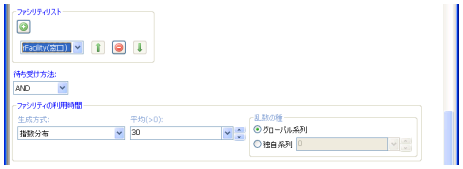


図 20:「窓口利用」の編集 (2)

並列処理で「並列処理」を選択し、並列数を「Inf」にします。並列数はお客が同時に窓口を利用できる数を表します。「Inf」を指定すると、窓口が無限に並んでいる事になります。お客は無限に並んでいる窓口に座り、担当者が来るのを待っているイメージです。また、「逐次処理」は、並列処理が 1 の場合と同義になります。(図 21)

他の項目はそのままにし、「OK」を押すと、部品編集画面が閉じます。

図 21: 「窓口利用」の編集 (3)

2.12　パラメータの編集

次にモデルパラメータを設定します。モデルパラメータとは、シミュレーションモデルの実行方法を定めるパラメータです。モデルメニューから「パラメータを編集する」を選ぶと (図 22)、パラメータ編集画面が表示されます。(図 23)

図 22: モデルメニュー

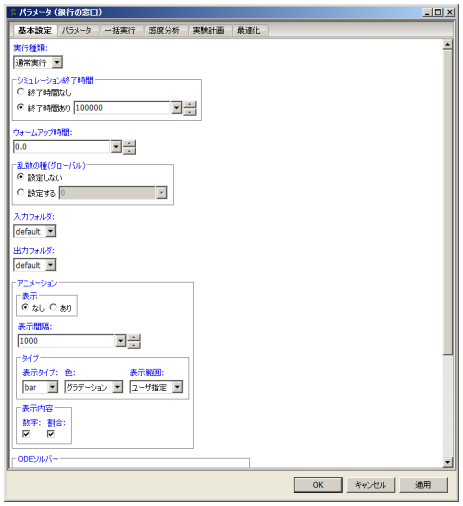


図 23: パラメータ編集画面

ここでは、シミュレーション終了時間を 100000 に設定します。シミュレーション終了時間にはシミュレーション内部時間を指定します。この例の場合はシミュレーション時間の単位は「秒」としていますが、シミュレーションによっては単位が「時間」、「日」の場合もあります。シミュレーション終了

時間を長くすればよりシミュレーションの分析を正確に行えますが、シミュレーションに要する実時間も長くなります。

他の項目はそのままにし、「OK」を押すと、パラメータ編集画面が閉じます。

2.13　モデルの実行

シミュレーションを実行する準備が整いましたので、シミュレーションを実行してみます。モデルメニューから「モデルを開始する」を選ぶと (図 22)、モデル実行画面が表示されます。(図 24) メーターはシミュレーションの進捗状況を示しています。

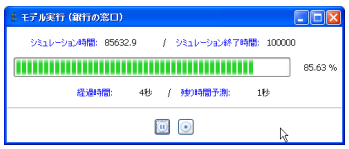


図 24: モデル実行画面

2.14　サマリの表示

モデルメニューから「サマリを表示する」を選ぶと (図 22)、サマリ画面が表示されます。(図 25)

図 25: サマリ画面

ファシリティ「窓口」の基本統計量が表示されます。「要求待ち」は「窓口」に平均でどのぐらいの人が並んでいるかを表しています。シミュレーションごとに結果は異なりますが、平均でおおよそ 0.5 人近辺の値を示していると思います。

次に窓口に平均でどれくらいの時間並んでいるかを調べます。もう一度「窓口利用」部品をダブルクリックして編集画面を開いて、一番下の「ファシリティ利用の記録」で「モニター」を選択し (図 26)、「OK」を押し編集画面を閉じて下さい。

モデルメニューから「モデルを開始する」を選び、再実行して下さい。もう一度モデルメニューから「サマリを表示する」を選ぶと、サマリ画面が表示されます。(図 27)

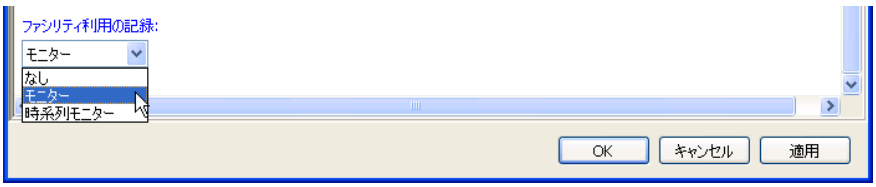


図 26: /ファシリティの記録

図 27: サマリ画面 (2)

「窓口利用-ファシリティの記録」が加わっています。「待ち時間」がファシリティの利用に要した時間を表します。シミュレーションごとに結果は異なりますが、平均でおおよそ 30.0 秒近辺の値を示していると思います。

この結果から、

- お客の到着間隔は平均 60 秒の指数分布に従う。
- 窓口のサービス時間は平均 30 秒の指数分布に従う。

であれば、待ち行列はおおよそ 0.5 人程度であり、待ち時間はおおよそ 30.0 秒程度になる事が分かりました。

2.15　データの表示

次に実際のデータを見てみます。ブラウザパネルのワークスペースタブを開いて下さい。「窓口利用-ファシリティの記録」で右クリックするとメニューが表示されます。(図 28)

「データを表示する」を選ぶと、生データが表示されます。(図 29)

今の設定では最大待ち時間を設定していないため「待ち受け」列には全て「use」が表示されています。「待ち受け時間」列には、お客が窓口の待ち行列に並んでいる時間 (待ち行列に並び始めてから窓口サービスを受けられるまでの時間) が表示されています。

2.16　ヒストグラムの表示

「待ち受け時間」列を見ただけでは、特徴が分かりませんでした。サマリ画面から待ち時間の平均は 30.0 秒程度である事は分かっていますが、どんな

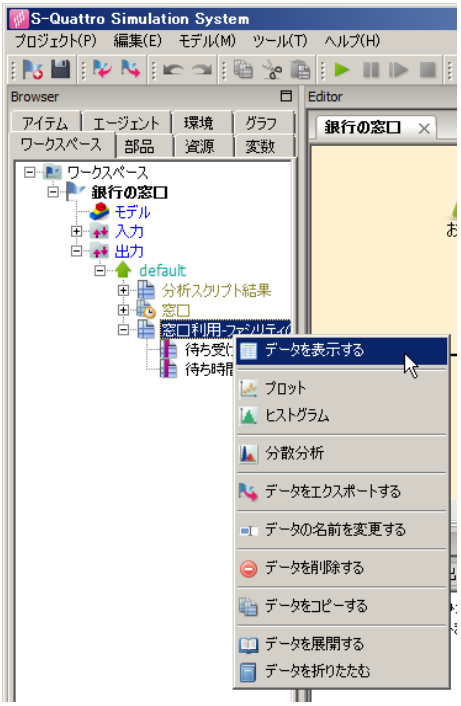
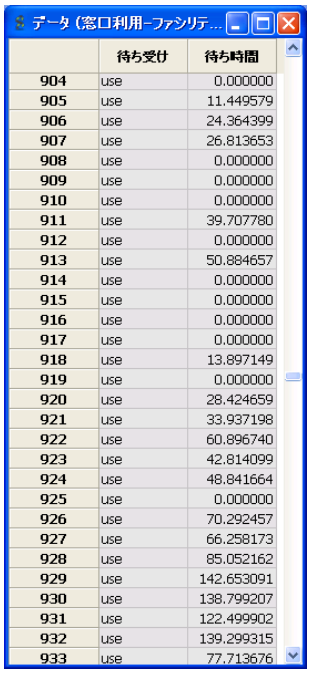


図 28: モニターメニュー



	待ち受け	待ち時間
904	use	0.000000
905	use	11.449579
906	use	24.364399
907	use	26.813653
908	use	0.000000
909	use	0.000000
910	use	0.000000
911	use	39.707780
912	use	0.000000
913	use	50.884657
914	use	0.000000
915	use	0.000000
916	use	0.000000
917	use	0.000000
918	use	13.897149
919	use	0.000000
920	use	28.424659
921	use	33.937198
922	use	60.896740
923	use	42.814099
924	use	48.841664
925	use	0.000000
926	use	70.292457
927	use	66.258173
928	use	85.052162
929	use	142.653091
930	use	138.799207
931	use	122.499902
932	use	139.299315
933	use	77.713676

図 29: データ表示

ばらつきをしているのかをヒストグラムで確認してみます。

「窓口利用-ファシリティの記録」で右クリックしたメニュー (図 28 で、「ヒストグラム」を選ぶと、ヒストグラムが表示されます。(図 30)

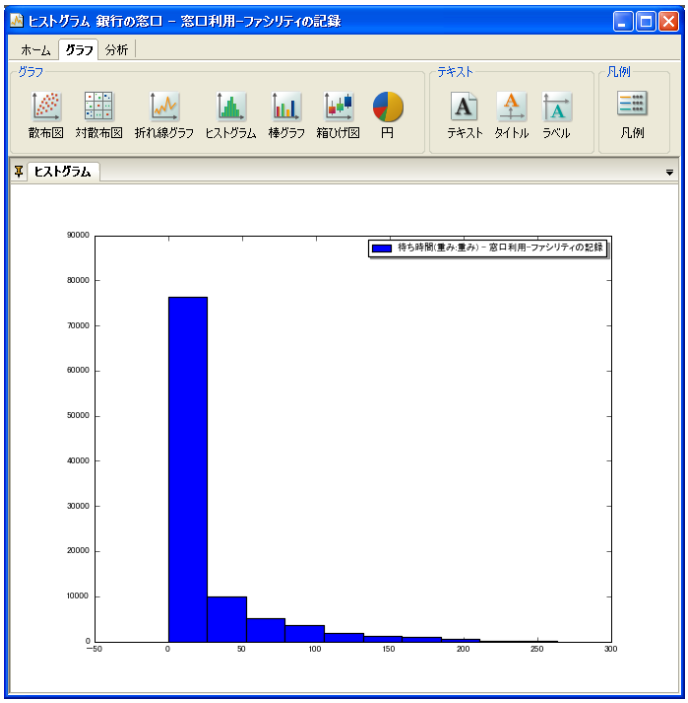


図 30: ヒストグラム

右にテールを引く分布である事が読み取れると思います。

2.17　時系列プロットの表示

次に窓口に並ぶ人数の時系列変化を確認してみます。

「窓口」で右クリックしたメニューから「時系列プロット」を選んで下さい。時系列変化が表示されます。(図 31)

平均待ち行列数は 0.5 人程度である事が分かっていますが、窓口の混雑状況にかかわらずお客はランダムに到着するため、待ち行列数がランダムに上昇、下降している様子が分かります。

2.18　リアルタイムグラフ

S4 Simulation System では、実行結果に対するグラフをリアルタイムに描画させる事も出来ます。ブラウザパネルの「グラフ」タブを開いて、「リアルタイムグラフ」をモデル上に配置します。(図 32)

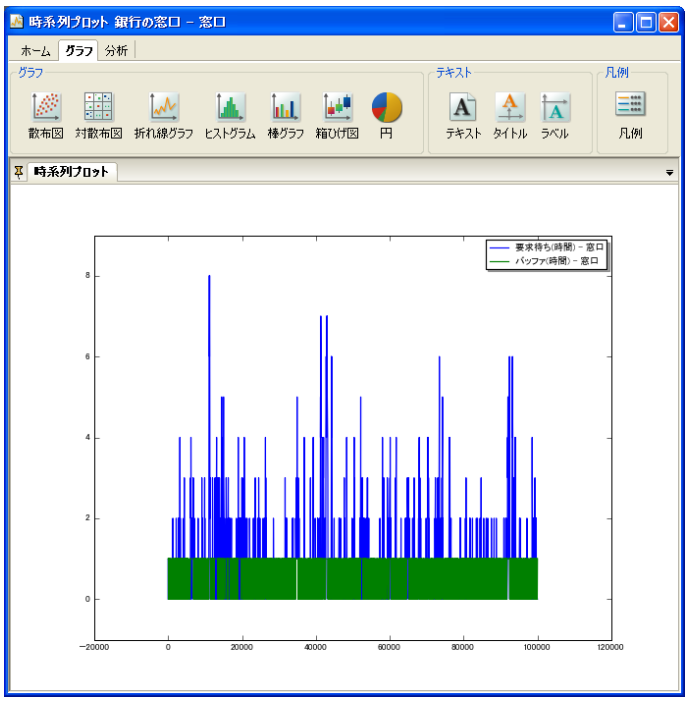


図 31: 時系列プロット

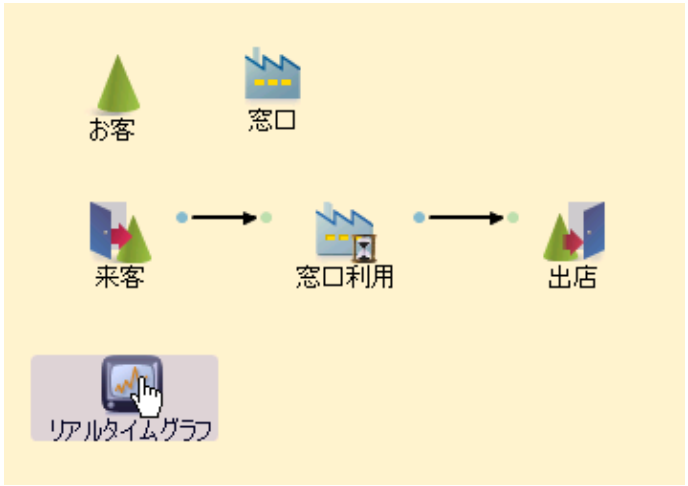


図 32: リアルタイムグラフの配置

「リアルタイムグラフ」をダブルクリックすると、グラフ作成画面が表示されます。グラフ作成画面上部の「ヒストグラム」をクリックすると、グラフ内にヒストグラムが配置され、その設定画面が表示されます。(図 33)

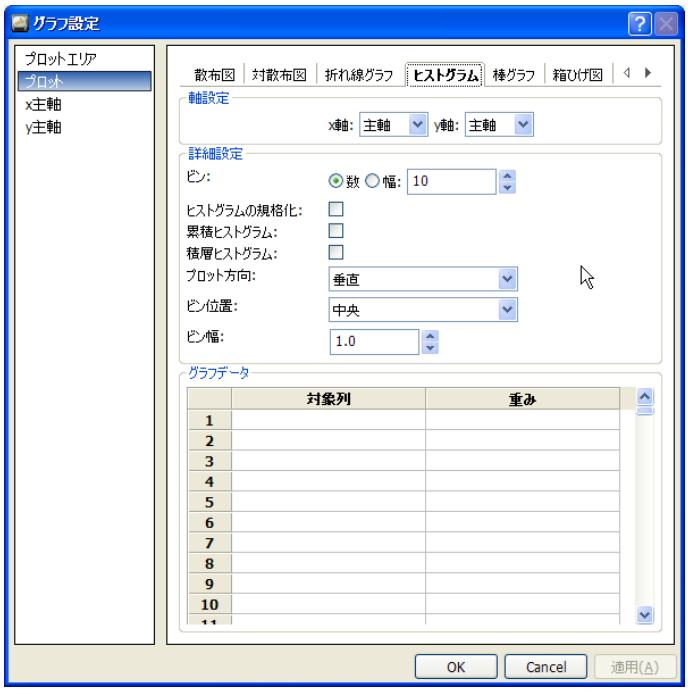


図 33: ヒストグラムの配置

グラフデータの 1 行目の「x」に、「窓口利用-ファシリティ」の中の「待ち時間」を設定します。(図 34)

グラフグループ設定の「OK」をクリックし、「銀行の窓口-リアルタイムグラフ」の右上の「X」をクリックします。モデルメニューから「モデルを開始する」を選び、実行すると、先程確認した待ち時間の分布がリアルタイムで表示されます。

2.19　記録部品の配置

記録部品を用いて来店者と退店者の数を時系列に記録する事で、直接的に窓口の混雑具合を確認する事が出来ます。来店者と退店者の記録には、「銀行の窓口」モデルに記録部品を以下のように配置し名前を編集します。

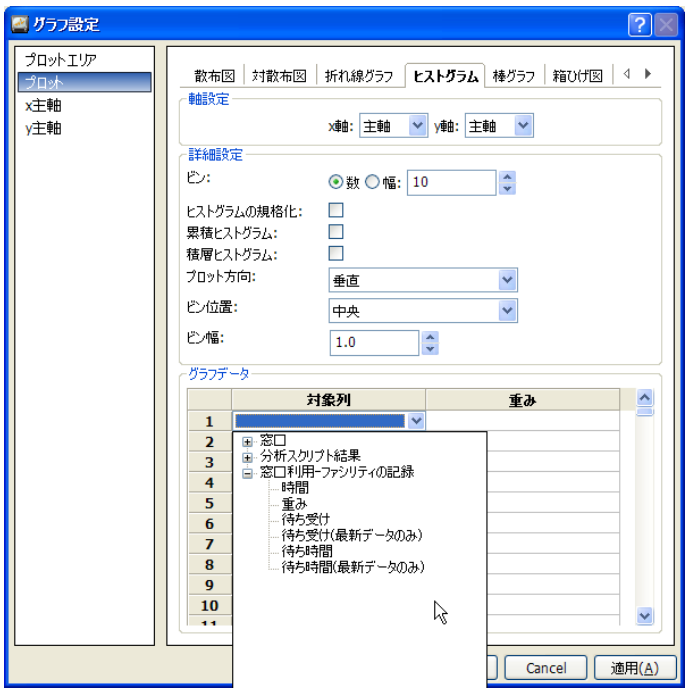


図 34: 列の設定

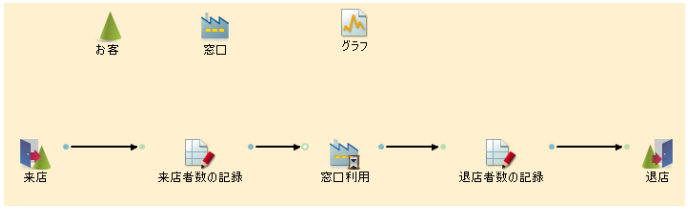


図 35: 記録部品の配置

2.20　記録部品の編集

来店者数の記録をダブルクリックして編集画面を開き、モニター種類を時系列モニターにします。また、記録内容の列名を来店、列の型を実数、記録方法を累積カウントにします。

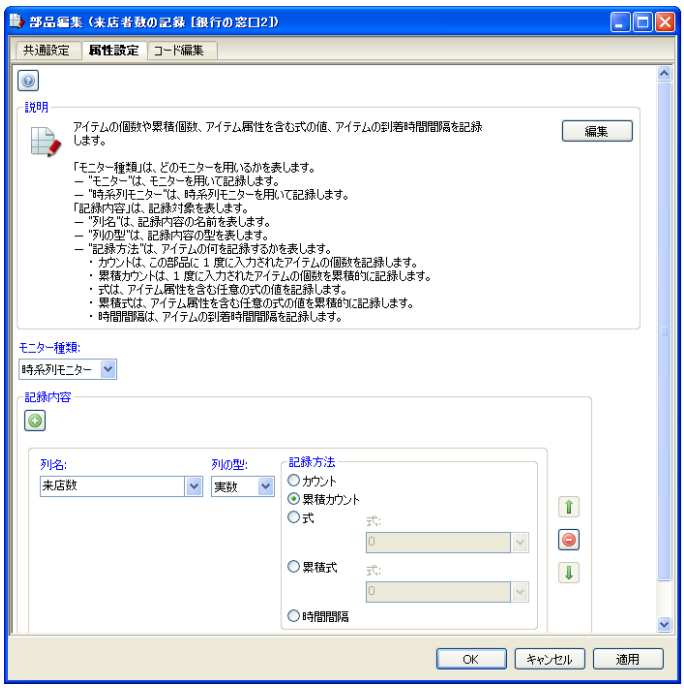


図 36: 来店者数の記録

退店者数の記録も同様に、モニター種類を時系列モニターにします。また、記録内容の列名を退店数、列の型を実数、記録方法を累積カウントにします。

図 37: 退店者数の記録

2.21　窓口の混雑具合の比較

まずは、窓口が 1 つの場合の混雑具合を確認します。「銀行の窓口」モデルのファシリティ「窓口」の「同時利用容量」を 1 にします。また今回は、窓口利用のファシリティの利用時間の平均を 50 に、シミュレーション終了時間を 5000 秒に変更します。その他のパラメータは、そのままにしておきます。パラメータを変更したらモデルを実行します。実行後、来店者数と退店者数のグラフを描きます。グラフを描くには、グラフ部品を配置してダブルクリックしてグラフ編集画面を開きます。グラフ編集画面が開いたら、折れ線グラフを選択しグラフデータ欄の 1 行目の x 軸に来店者数の記録の時間、y 軸に来店者数の記録の来店数を入力し、2 行目の x 軸に退店者数の記録の時間、y 軸に退店者数の記録の退店数を入力します。

「同時利用容量」は、窓口の担当者の人数に相当します。今、並列処理数には「Inf」が設定されているため、この場合は、無限に並んでいる窓口に対して、担当者は 1 人が順番に対応する事になります。

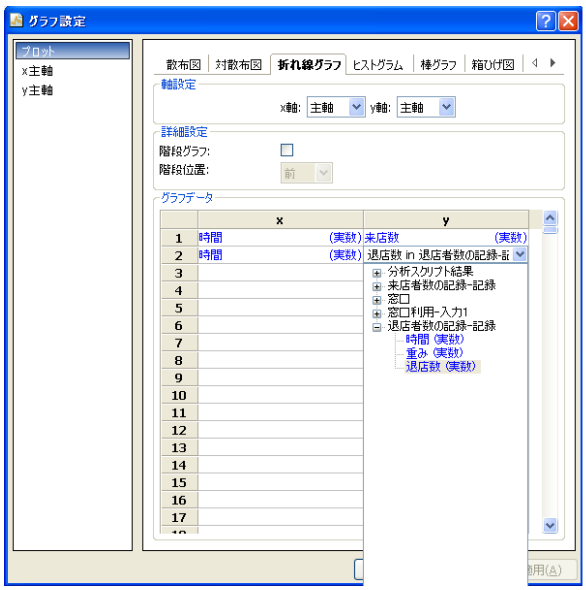


図 38: グラフの設定

グラフが描かれたら凡例を表示させるために、グラフタブにある凡例をクリックします。また、軸に名前を付ける為にラベルをクリックして x 軸に時間、y 軸に累積人数と入力します。このようにして作成されたグラフは以下のようになります。

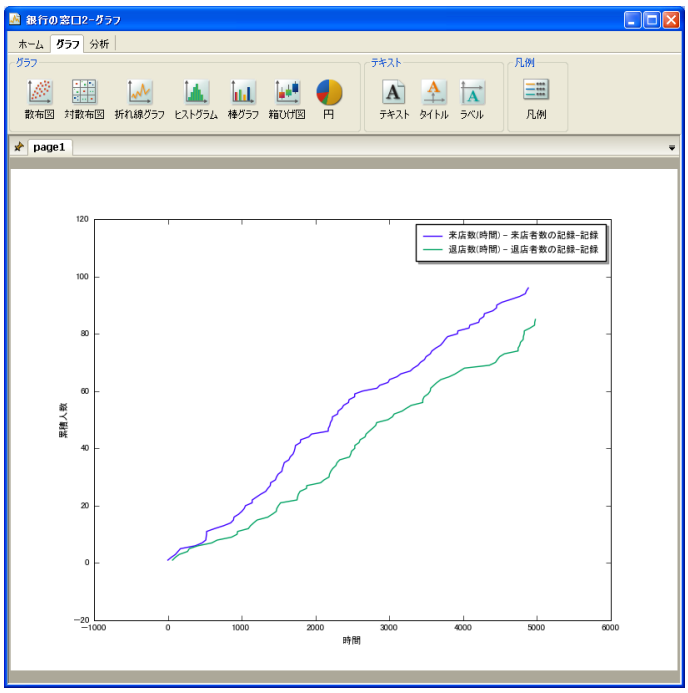


図 39: 窓口数 1 の滞留人数の様子

来店者数と退店者数に挟まれた面積が銀行に滞留している人数になります。パラメータ (この場合は窓口の数) を調整して窓口がなるべく混雑しないようにすると、この面積は小さくなります。

次に窓口の担当者の人数を 2 人にした場合を考えます。今作成したモデルのファシリティ「窓口」の「同時利用容量」を 2 にしてもう一度モデルを実行します。実行が終わったら、先程作成したグラフ部品をダブルクリックすると窓口の担当者の人数が 2 人の場合のグラフが表示されます。

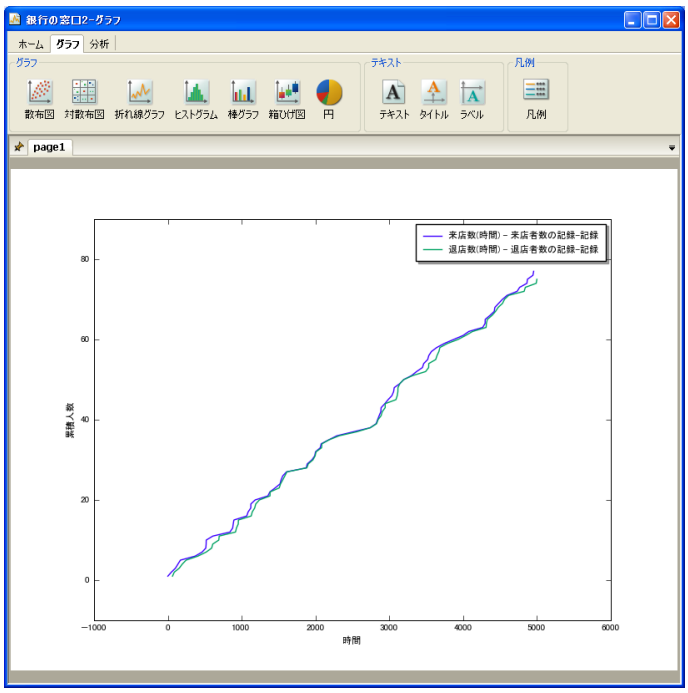


図 40: 窓口の担当者数 2 の滞留人数の様子

窓口の担当者の人数を 2 人にすると、来店者数と退店者数に挟まれた面積はほぼ 0 になり滞留人数が少なくなった事がわかります。

2.22　システムの挙動変化の分析

このようにシミュレーションでは条件を様々に変えてシステムの挙動の変化を調べる必要があります。例えば、担当者の人数の他に来店数 (間隔) が変わった場合や、窓口でのサービス時間を変えた場合に、窓口に並ぶ人数や並ぶ時間はどのように変わるのでしょうか。S4 Simulation System では以下のようにパラメータを動かして結果を分析することがきます。

- 「来店」部品の「生成時間」は来店間隔を表します。来店間隔を変えると、どうなるでしょうか。
- 「窓口利用」部品の「ファシリティの利用時間」は窓口のサービス時間を表します。サービス時間を換えると、どうなるでしょうか。
- 来店間隔の平均が窓口利用時間の平均より小さいとどうなるでしょうか。

3　銀行の窓口モデル応用

先程のモデルは窓口が一つの単純なモデルでした。ここでは窓口の種類が3種類ある少し複雑なモデルを考えます。

3.1　モデル

モデルの概要は以下の通りです。

- 銀行には「預金受付」、「融資申し込み」、「投資相談」の 3 つのサービスがある
- 各サービスはそれぞれ専用の窓口で処理を行う
- 「お客」は 1F と 2F から来店しそれぞれ目的に応じて各窓口に並ぶ
- 1F から来店する「お客」は平均 20 秒の指数分布に従う
- 2F から来店する「お客」は平均 240 秒の指数分布に従う
- 1F から来店したお客の 90 %が預金、8 % が融資、2 %が投資相談に行く
- 2F の入り口は「投資相談窓口」専用である
- 各窓口でサービスを受けたお客は、他の窓口には行かずそのまま退店する
- 各窓口には担当者が一人ずついる
- 行列に並べる人数はそれぞれ預金が 30 人、融資が 10 人, 投資相談が 10 人までである
- サービス提供時間は平均預金が 25 秒、融資申し込みが 120 秒、投資相談が 145 秒の指数分布に従う

3.2　部品の配置

「銀行の窓口モデル」で作成したように、以下のように部品を配置して名前を変更します。

図 41: 銀行の応用モデルの部品の配置

3.3　リンクの作成

1F から来店したお客は預金、融資、投資相談に行くので次のようにリンクを作成します。

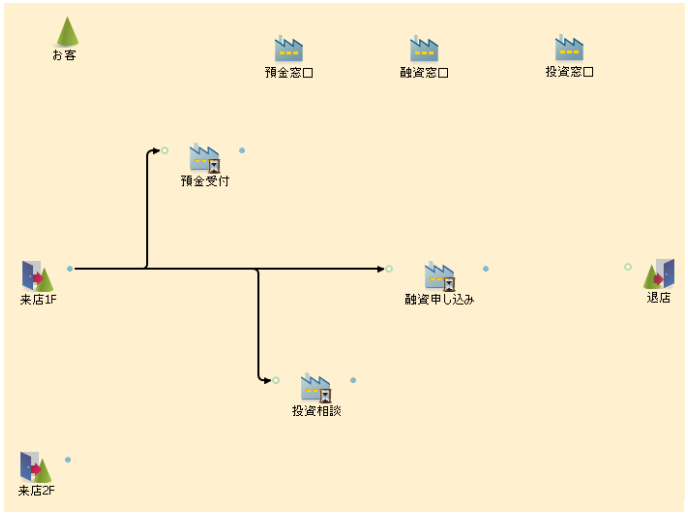


図 42: 1F のお客の来店

2F の入り口は「投資相談窓口」専用なので次のようにリンクを作成します。

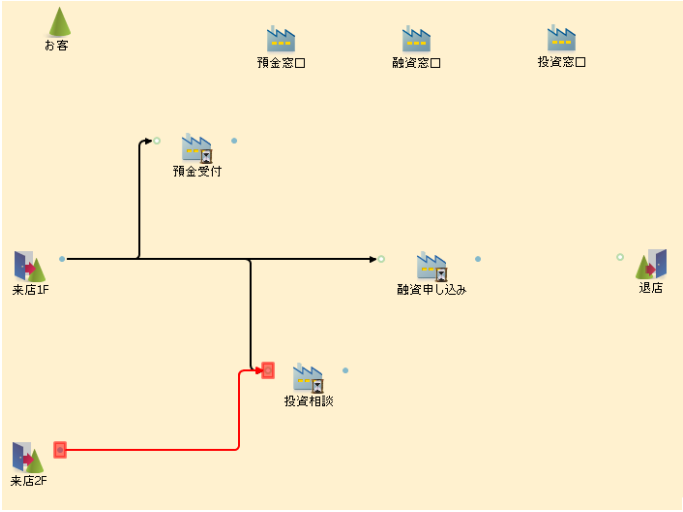


図 43: 2F のお客の来店

サービスを受けたお客はそのまま退店するので次のようにリンクを作成します。

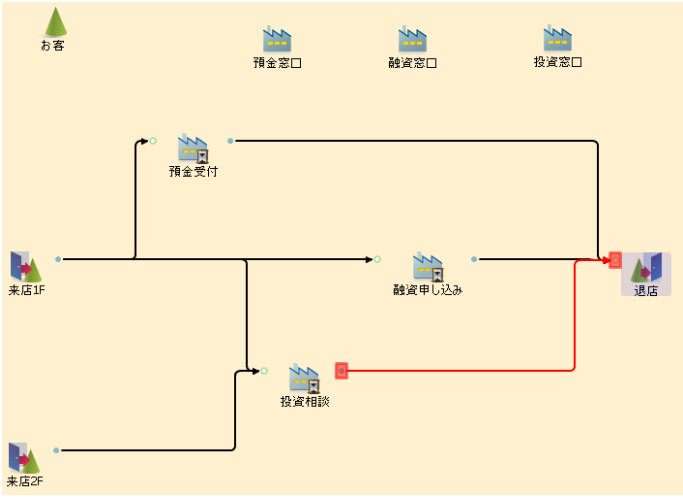


図 44: お客の退店

3.4　来店の編集

「来店 1F」部品をダブルクリックして編集画面を開きます。編集画面が表示されたら 1F から来店する「お客」は平均 20 秒の指数分布に従うので、生成時間の生成方式を指数分布にし、平均に 20 を設定します。

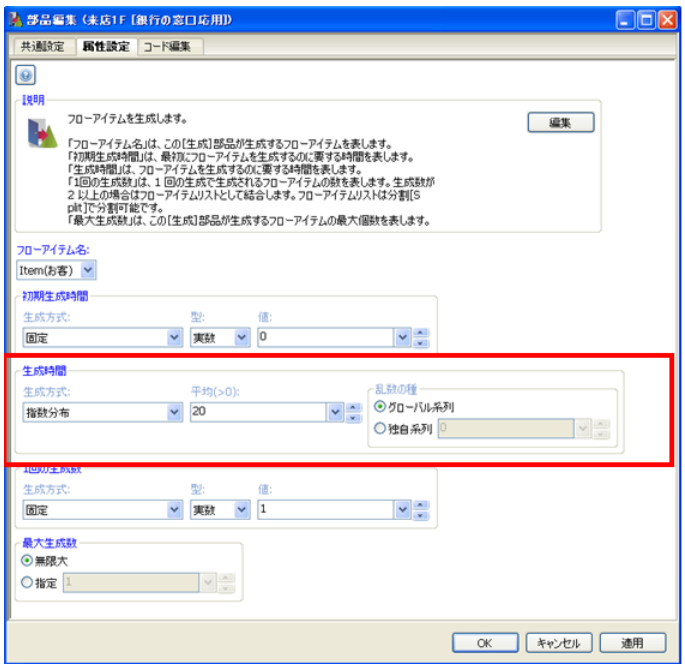


図 45:「来店 1F」の編集

また、1F から来店したお客の 90 %が預金、8 % が融資、2 %が投資相談に行くので、共通設定の出力ポートの選択方式でセレクタをランダム (重み付き) にし、重み表の預金受付に 90、融資申し込みに 8、投資相談に 2 を入力します。重み表の行を追加するには、表の上で右クリックしてメニュを表示します。

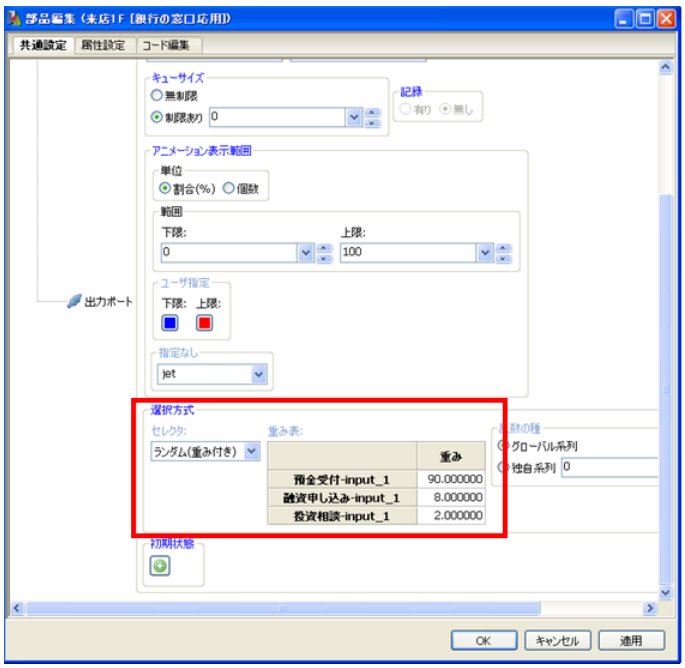


図 46:「来店 1F」の出力ポートの編集

同様に 2F から来店する「お客」は平均 240 秒の指数分布に従うので、来店 2F 部品の編集画面を編集します。

3.5　窓口の編集

「預金窓口」部品をダブルクリックして編集画面を開きます。窓口にいる担当者の人数は 1 人なので、同時利用容量を 1 にします。「融資窓口」、「投資窓口」も同様に編集します。

3.6　窓口利用の編集

「預金受付」部品をダブルクリックして編集画面を開きます。預金受付では、預金窓口を利用するのでファシリティリストに預金窓口を指定します。また、サービスの提供にかかる時間は平均 25 秒の指数分布に従うので、ファシリティの利用時間の生成方式を指数分布にし、平均に 25 を設定します。今回は担当者数と同じ数だけ窓口を用意しますので、並列処理にチェックをし、並列数に 1 を設定します。

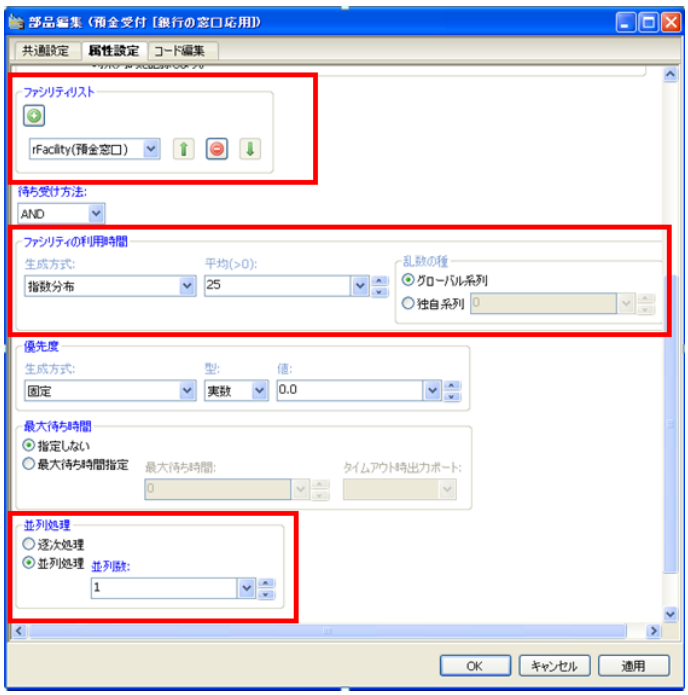


図 47:「預金受付」の編集

預金受付の行列に並べる人数は 30 人なので、共通設定のキューサイズを制限ありにし 30 を入力します。

今回は、窓口の数を担当者の人数と同じにしているので、お客が並ぶ場所を作らなければなりません。入力のキューサイズに無制限、あるいは制限ありで、数を指定すると窓口の前に行列を作ることができます。つまり、お客は窓口に座って待つのではなく、窓口の前に行列を作ることになります。

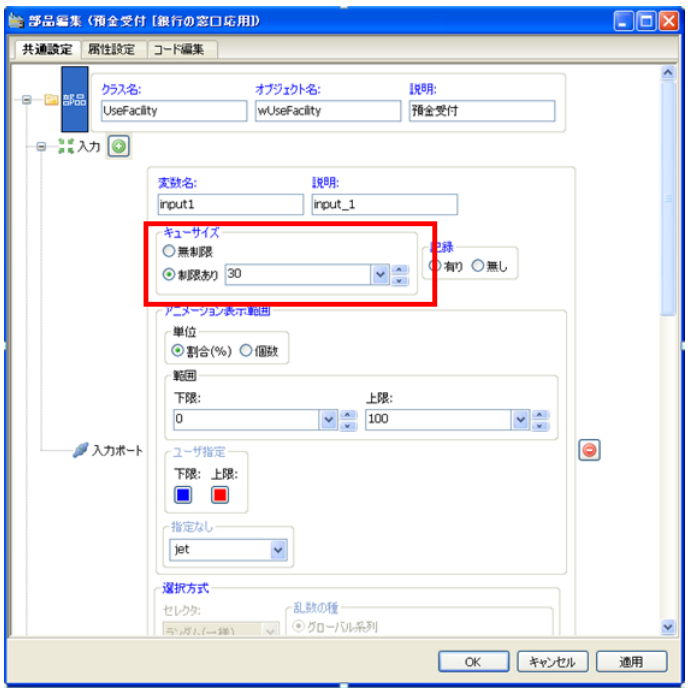


図 48:「預金受付」のキューの編集

融資申し込みも同様に、ファシリティリストに融資窓口を、ファシリティの利用時間の生成方法を指数分布にし、平均に 120 を設定します。また、並列数に 1 を設定します。さらに共通設定のキューサイズを制限ありにし 10 にします。投資申し込みについても、ファシリティリストに投資窓口を、ファシリティの利用時間の生成方法を指数分布にし、平均に 145 を設定します。また、並列数に 1 を設定します。さらに共通設定のキューサイズを制限ありにし 10 にします。

3.7　シミュレーション終了時間の設定

パラメータ編集画面を開いて、シミュレーション時間に 500000 を入力します。

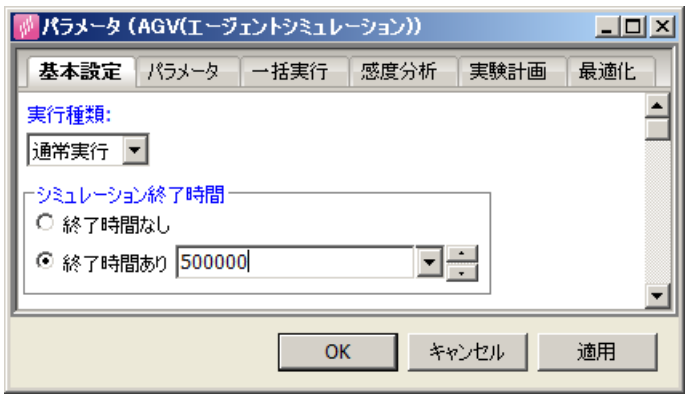


図 49: シミュレーション終了時間の設定

3.8　アニメーション機能

どの窓口に行列が出来るかを把握するには、アニメーション機能を用いると便利です。アニメーション機能とは、シミュレーション実行中に待ち行列の長さを色で知らせる機能です。これによりシミュレーション中にどこに待ち行列が発生しやすいかを視覚的に確認する事が出来ます。

3.9　アニメーションパラメータ編集

アニメーションを利用するには、パラメータ編集画面でアニメーション表示をありにします。アニメーションの表示速度はスライドバーによって変更できます。

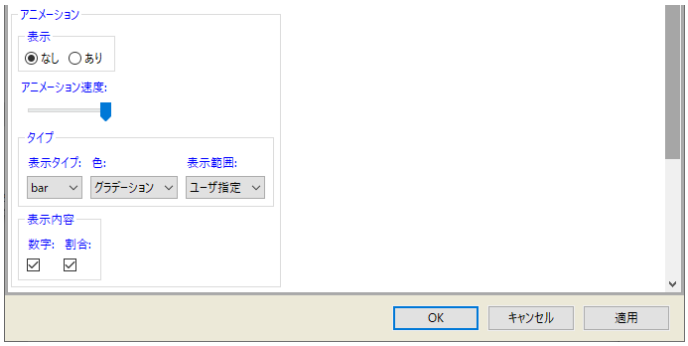


図 50: アニメーションの設定

3.10　アニメーションの実行

モデルを実行すると、待ち行列が発生している箇所に行列の長さと行列の上限数に対する割合が数値とグラデーションで表示されます。この例では、「預金受付」と「投資相談」に待ち行列が発生しやすい事が分かります。また、預金窓口、融資窓口、投資窓口の表示はそれぞれの利用率を表しています。効率よく利用出来ていればここの値は常に高くなります。

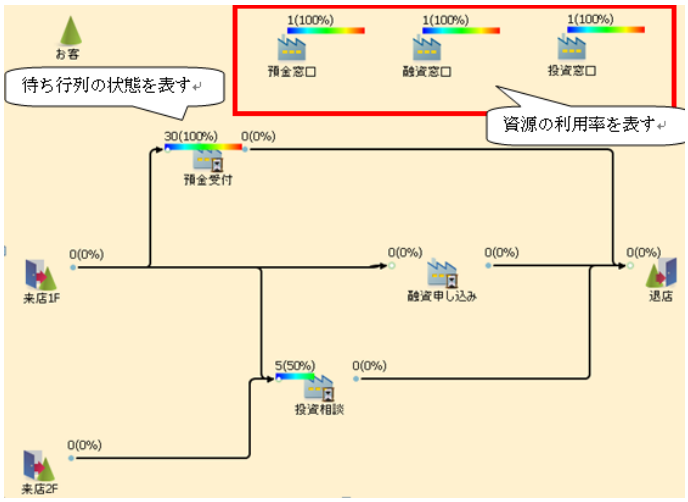


図 51: アニメーションの実行

4　製造工程モデル

製品の製造工程を考えます。ここで考える製造工程は、製品の種類が 1 種類であり、複数の工程があり、工程ごとに使用する装置が異なる場合を考えます。また、それぞれの工程で使用する装置は複数あり、それらは同時に利用することが可能である場合を想定します。

4.1　製造工程モデル

製品組み立てモデルは、パーツから製品を組み立てるモデルです。(図 52)

- パーツは無限に供給されているとする。
- 製造工程は全部で 4 工程あります。
- 工程ごとに使用する装置が異なります。
- 装置は複数あります。

- 装置と装置の間にはバッファーがあります。
- 装置に続くバッファーが一杯な場合には装置は新たなパーツの処理を行えません。
- 装置の処理時間はランダムです。

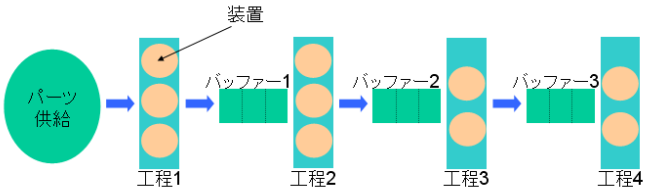


図 52: 製造工程モデル

具体的にはそれぞれの装置の同時処理数と処理時間は以下の通りとします。

装置	同時処理数	処理時間
装置 1	3	平均 0.33333 の指数分布
装置 2	2	平均 0.5 の指数分布
装置 3	2	平均 0.2 の指数分布
装置 4	3	平均 0.25 の指数分布

また、バッファーの大きさは以下の通りとします。

バッファー	場所	サイズ
バッファー 1	装置 1 と装置 2 の間	3
バッファー 2	装置 2 と装置 3 の間	1
バッファー 3	装置 3 と装置 4 の間	2

4.2　プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「製造工程」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。次に、製造工程モデルを作成していきます。

4.3　フローアイテムの配置

製造装置モデルでは、それぞれの装置で加工されるパーツをアイテムで表現します。ブラウザパネルで「アイテムタブ」を選択してください。アイテムタブの「アイテム」をドラッグし、モデル編集画面にドロップしてください。アイテムが配置されます。アイテムはシミュレーション内に流れるパーツを表しますので、名前を「パーツ」と変更します。

4.4　生成部品の配置

製造工程では、まずパーツが供給されなくてはなりません。パーツの供給は「生成」部品で表現します。ブラウザパネルの「部品」タブを選択し、生成部品をモデル編集画面に配置します。この部品はパーツの供給を表しますので「パーツ供給」という名前に変更します。

4.5　ファシリティの配置

装置がある部品を加工している間、ほかの部品は製造装置を利用することができません。これは資源の排他的な利用になります。そこで、装置を「ファシリティ」で表現することにします。

ブラウザパネルで「資源」タブを選択し、資源タブから「ファシリティ」をモデル編集画面に配置します。製造工程は 4 工程あり、それぞれ装置が異なります。そこで、ファシリティを 4 つモデル編集画面に配置します。このファシリティは装置を表すので、それぞれ「装置 1」、「装置 2」、「装置 3」、「装置 4」と名前を変更します。

4.6　ファシリティ利用部品の配置

実際に製造装置を排他的に利用する製造工程を「ファシリティ利用」部品で表現します。ブラウザパネルで「部品」タブを選択し、「ファシリティ利用」部品をモデル編集画面に配置します。製造工程は 4 工程あるので、それぞれに対応して 4 つ配置します。それぞれ「工程 1」、「工程 2」、「工程 3」、「工程 4」と名前を変更します。

4.7　終端部品の配置

完成した製品の出力を「終端」部品で表します。ブラウザパネルで「部品」タブを選択し、「終端」部品をモデル編集画面に配置します。終端部品は製品の出力を表すので「出荷」と名前を変更します。

4.8　リンクの作成

製造工程におけるパーツの流れをリンクで表現します。「パーツ供給」部品と「工程 1」部品にリンクを作成します。「工程 1」部品と「工程 2」部品にもリンクを作成します。「工程 2」部品と「工程 3」部品にもリンクを作成します。「工程 3」部品と「工程 4」部品にもリンクを作成します。「工程 4」部品と「出荷」部品にもリンクを作成します。

出来上がったプロジェクトは以下のようになります。(図 53)

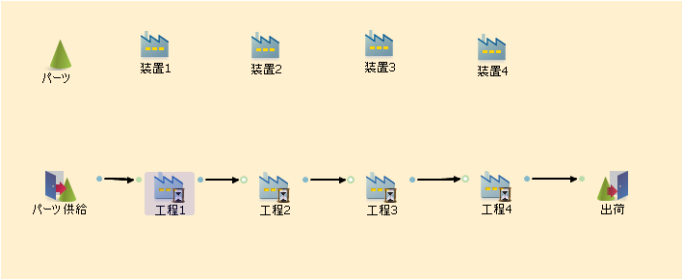


図 53: 製造工程モデル編集画面

4.9　パーツ供給の編集

パーツ供給の振る舞いを指定するために、「パーツ供給」を編集します。「パーツ供給」部品をダブルクリックすると部品編集画面が表示されます。「パーツ供給」部品では、パーツの供給を表現します。この製造工程モデルではパーツの供給は無限大ですので、それを表現するために

パラメータ名	値
フローアイテム名	Item(パーツ)
生成時間	0
生成最大数	無限大

と指定します。(図 54) これにより、パーツを無限に生成するという指定になります。

この指定では大量のパーツを生成するように見えますが、S4 Simulation System では、入力、出力ポートのキューサイズをコントロールすることにより、部品の動作を制限することができます。この場合、「パーツ供給」部品につながる部品 (工程 1) の入力ポートのサイズを制限することで、実際には「パーツ供給」部品は、「工程 1」に空きがある場合のみ、パーツを供給するという動作になります。

図 54: パーツ供給部品 編集画面

4.10　装置の編集

装置の振る舞いを指定するために、「装置」を編集します。「装置」をダブルクリックすると資源編集画面が表示されます。この製造工程モデルでは、それぞれの装置での同時処理数が「装置 1」が 3、「装置 2」が 2、「装置 3」が 2、「装置 4」が 3 です。同時処理数の指定は同時利用量を設定することで行います。それぞれの「装置」の「同時利用量」にこの値を指定します。(図 55)

また、「共通設定」タブをクリックし、「オブジェクト名」を変更します。(図 56)

装置	同時利用量	オブジェクト名
装置 1	3	WS1
装置 2	2	WS2
装置 3	2	WS3
装置 4	3	WS4

4.11　工程の編集

装置の利用の仕方を指定するために、「工程」部品を編集します。「工程」部品をダブルクリックすると部品編集画面が表示されます。この製造工程モ

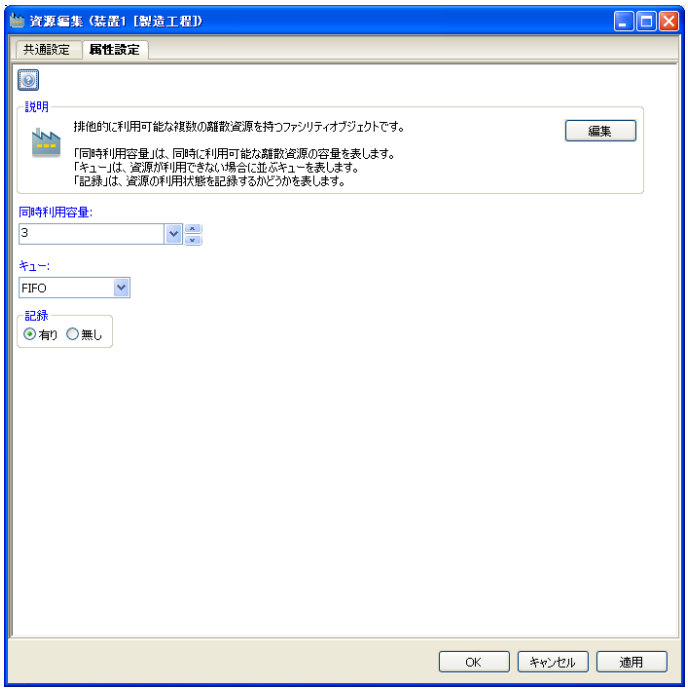


図 55: 装置資源 編集画面

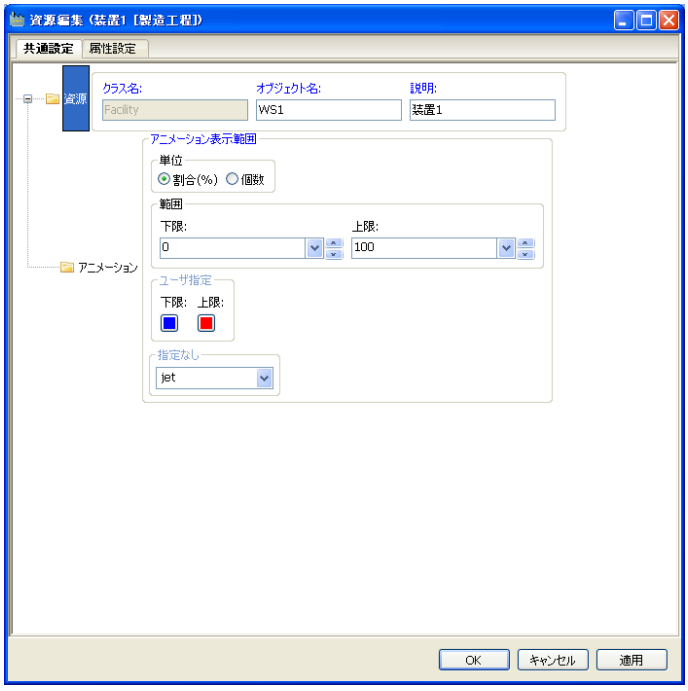


図 56: 装置資源 共通設定画面

デルでは、それぞれの装置を同時利用します。それを表現するために、属性設定タブで以下のように指定します。(図 57)

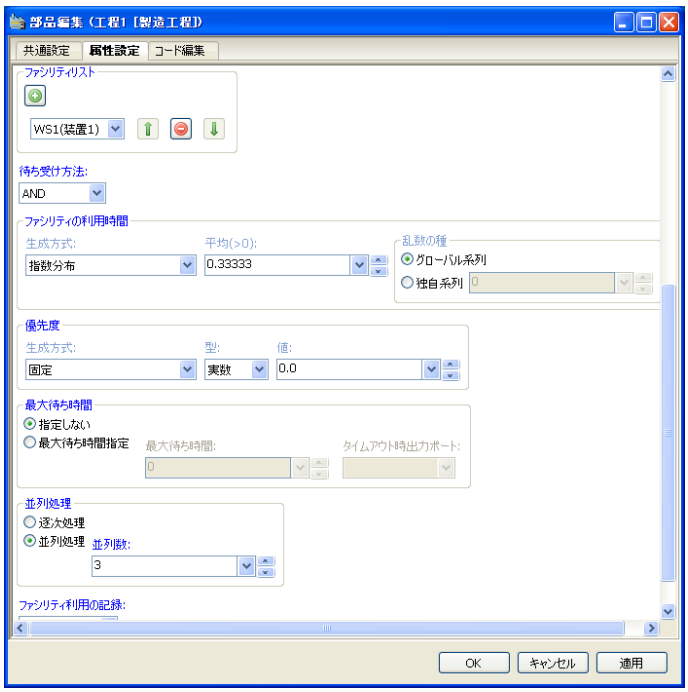


図 57: 工程部品 編集画面

工程	ファシリティリスト	利用時間	並列処理数
工程 1	WS1(装置 1)	平均 0.333 の指数分布	3
工程 2	WS2(装置 2)	平均 0.5 の指数分布	2
工程 3	WS3(装置 3)	平均 0.2 の指数分布	2
工程 4	WS4(装置 4)	平均 0.25 の指数分布	3

「工程」での並列数とは、装置に対してその利用を同時に要求する数となります。「装置」は「工程」から要求を受け、装置が空いていればその要求に従い装置を指定された時間占有し、その後、解放します。装置が空いていない場合には、装置が空くのを待ちます。いくつまで待てるかがバッファーのサイズということになります。これを入力ポートのキューサイズで指定します。「共通設定」タブの「入力」の「キューサイズ」を制限ありとし、それぞれの工程でのキューサイズを以下のように指定します。(図 58)

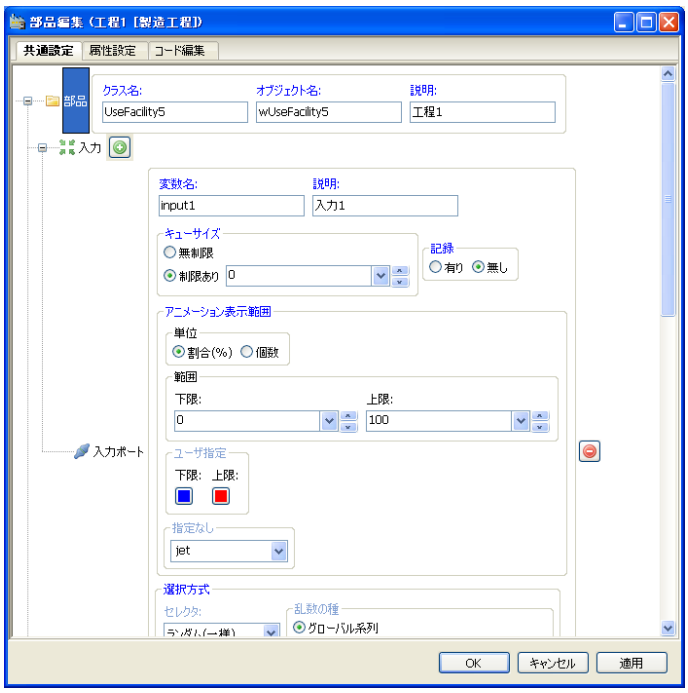


図 58: 工程部品 共通設定画面

装置	キューサイズ
装置 1	0
装置 2	3
装置 3	1
装置 4	2

4.12　出荷の編集

出荷される製品数をカウントするために「出荷」部品を編集します。「出荷」部品をダブルクリックすると部品編集画面が表示されます。「共通設定」タブをクリックし、入力の「記録」を「有り」にします。(図 58) これにより、「出荷」部品へ入力した部品の数をカウントします。

4.13　実行パラメータの編集

製造工程モデルにおける実行パラメータを編集します。「モデル」メニューの「パラメータを編集する」を選択し、パラメータ編集画面を開きます。「基本設定」タブで以下の基本的な情報を設定します。(図 60)

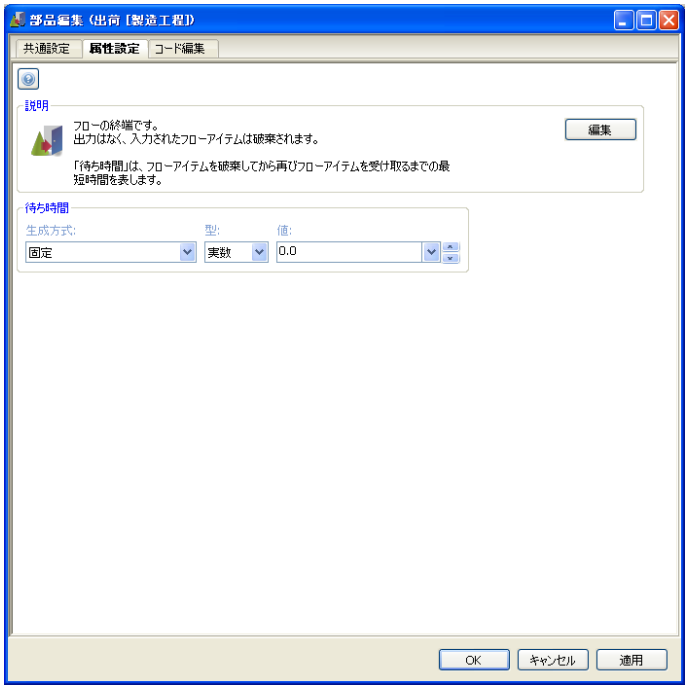


図 59: 出荷部品 編集画面

図 60: 実行パラメータ 編集画面

パラメータ名	値
シミュレーション時間	96
ウォームアップ時間	24

シミュレーション時間はシミュレーションを行う時間です。ウォームアップ時間はシステムをウォームアップする時間です。シミュレーション開始からウォームアップ時間まで、システムは実際のシミュレーションを行い状態変化しますが、その結果は保存されません。

4.14 目的関数

シミュレーションの評価関数を編集します。ここでは、装置数を M とし、バッファ数を B とし、製造される製品数を n とし、装置一つのコストを 250 万円、バッファ一つを 10 万円のコストがかかるとします。製品は一つ 20 万円で売れるとします。この時、評価関数は以下のようになります。

$$F(M,B,n) = 20 \times n - 250 \times M - 10 \times b$$

シミュレーションの評価関数は、分析スクリプトで設定することができます。「モデル」の「分析スクリプトを編集する」を選択し、分析スクリプト編集画面を開きます。objective の定義を以下のように書きます。

```
n=TimeMonitor(name=u"出荷-入力 1", basedir=self.outputDir)[u"追加待ち"].count()
m=3+2+2+3
b=3+1+2
return n*20-250*m-10*b
```

4.15 モデルの実行

「モデル」メニューから「モデルを開始する」を選択し、シミュレーションを開始します。Information ウィンドウに実行結果が表示されます。おおよそ 3500 万円 くらいの値になります。

5 最適化

シミュレーションは「What if」形式にさまざまに条件を変えながらその振る舞いを分析する手法です。しかし、この方法では調べられる範囲が限られ、システムを最適化しようとするには不十分です。ここでは、このようなシステムの振る舞いを最適にするパラメータを求めたい場合について考えてみます。

5.1　問題設定

最適化では最適化すべき目的関数（先の例ではシミュレーションの評価関数）とその際のパラメータを指定します。先ほどと同じように、装置数を M とし、バッファ数を B とし、製造される製品数を n とし、装置一つのコストを 250 万円、バッファ一つを 10 万円のコストがかかるとします。製品は一つ 20 万円で売れるとします。この時、目的関数は以下のようになります。

$$F(M,B,n) = 20 \times n - 250 \times M - 10 \times b$$

また、装置数は全体で最大 10 とし、バッファサイズはそれぞれ最大 10 までとします。よって、制約条件は以下のようになります。

$$\begin{aligned} M_1 + M_2 + M_3 + M_4 &\leq 10 \\ 1 \leq B_1 &\leq 10 \\ 1 \leq B_2 &\leq 10 \\ 1 \leq B_3 &\leq 10 \\ 1 \leq B_4 &\leq 10 \end{aligned}$$

5.2　パラメータ設定

製造工程モデルにおける最適化のためのパラメータを編集します。「モデル」メニューの「パラメータを編集する」を選択し、パラメータ編集画面を開きます。「パラメータ」タブでシミュレーションで用いる変数を定義します。変数左上の「+」ボタンを押してを追加します。変数名、生成方式、型、値は以下のように設定します。(図 65)

変数名	生成方式	型	値
WS1	固定	整数	3
WS2	固定	整数	2
WS3	固定	整数	2
WS4	固定	整数	3
Buffer1	固定	整数	3
Buffer2	固定	整数	1
Buffer3	固定	整数	2

5.3　部品の設定

部品や資源の動作をこれらのパラメータを用いて記述します。(図 62)
装置に関しては以下のようになります。

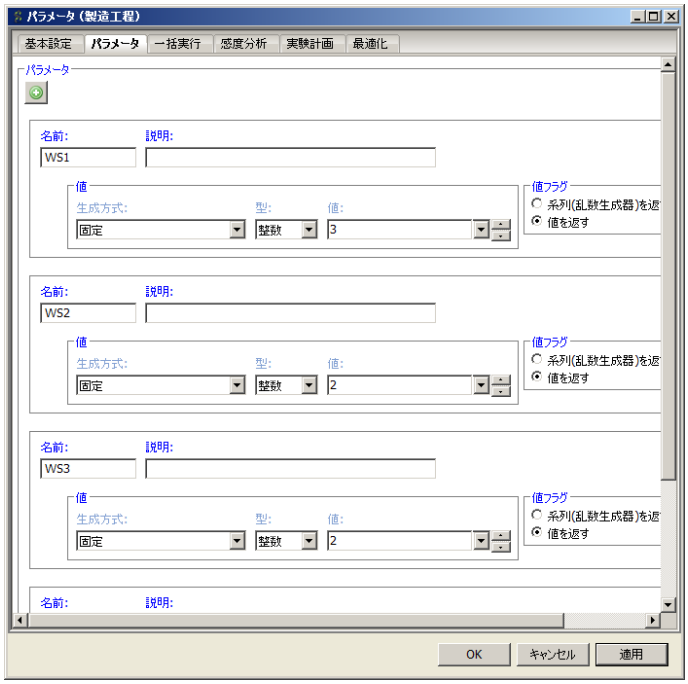


図 61: パラメータ 編集画面

装置	同時利用容量
装置 1	self.param.WS1
装置 2	self.param.WS2
装置 3	self.param.WS3
装置 4	self.param.WS4

工程に関しては以下のようになります。

装置	並列処理の並列数	入力キューサイズ
工程 1	param.WS1	0
工程 2	param.WS2	self.param.Buffer1
工程 3	param.WS3	self.param.Buffer2
工程 4	param.WS4	self.param.Buffer3

これにより、実行時にこれらのパラメータを用いてシミュレーションが行われるようになります。

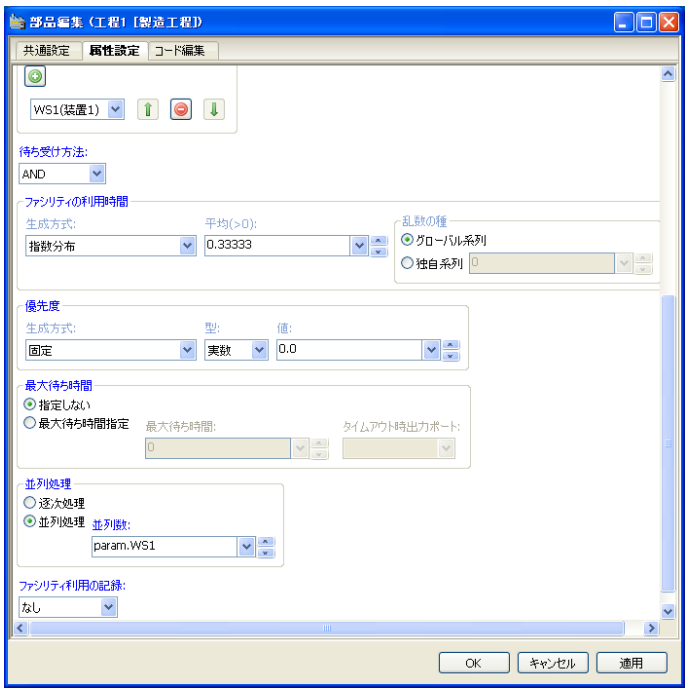


図 62: 装置 編集画面

5.4　目的関数

最適化の目的関数を編集します。先ほどの評価関数を、最適化のパラメータを使って書きなおします。「モデル」の「分析スクリプトを編集する」を選択し、分析スクリプト編集画面を開きます。objective の定義を以下のように書きます。(図 63)

```
n=TimeMonitor(name=u"出荷-入力 1", basedir=self.outputDir)[u"追加待ち"].count()
m=self.param.WS1+self.param.WS2+self.param.WS3+self.param.WS4
b=self.param.Buffer1+self.param.Buffer2+self.param.Buffer3
return n*20-250*m-10*b
```

このようにすることで、パラメータを変更した際に目的関数も変更されるようになります。

5.5　最適化設定

実行のオプションとして最適化実行を行う設定をします。「モデル」メニューの「パラメータを編集する」を選択し、パラメータ編集画面を開きます。

まず、「基本設定」タブの「実行種類」を「最適化」にします。

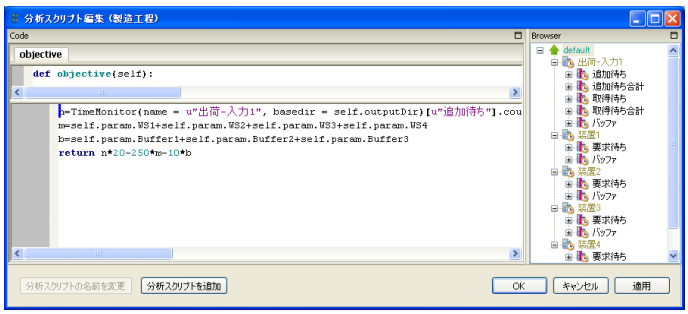


図 63: 分析スクリプト 編集画面

図 64: 基本設定タブ実行種類

次に、「最適化」タブで最適化に用いるパラメータを定義します。

「最適化設定」タブで最適化変数を「+」ボタンを押して追加します。パラメータ名には先ほど設定したパラメータをコンボリストから選択し、以下のように設定します。(図 65)

パラメータ名	型	下限	上限
WS1	整数	1	10
WS2	整数	1	10
WS3	整数	1	10
WS4	整数	1	10
Buffer1	整数	1	10
Buffer2	整数	1	10
Buffer3	整数	1	10

最適化のオプションは以下のように指定します。(図 66)

目的関数はデフォルトの self.objective のままで構いません。シミュレーションの結果は確率的に変わるため、目的関数の値は一意に求めることができません。そこで、シミュレーションの最適化においては、目的関数の期待値を計算します。期待値を計算するためのサンプル数がレプリケーションとなります。ここでは最大回数を指定することもできますし、目的関数の信頼区間で指定することもできます。ここでは、デフォルトのまま最小回数と最大回数を 5 で行います。

S4 Simulation System では、最適化は Dervative Free Optimization(DFO)

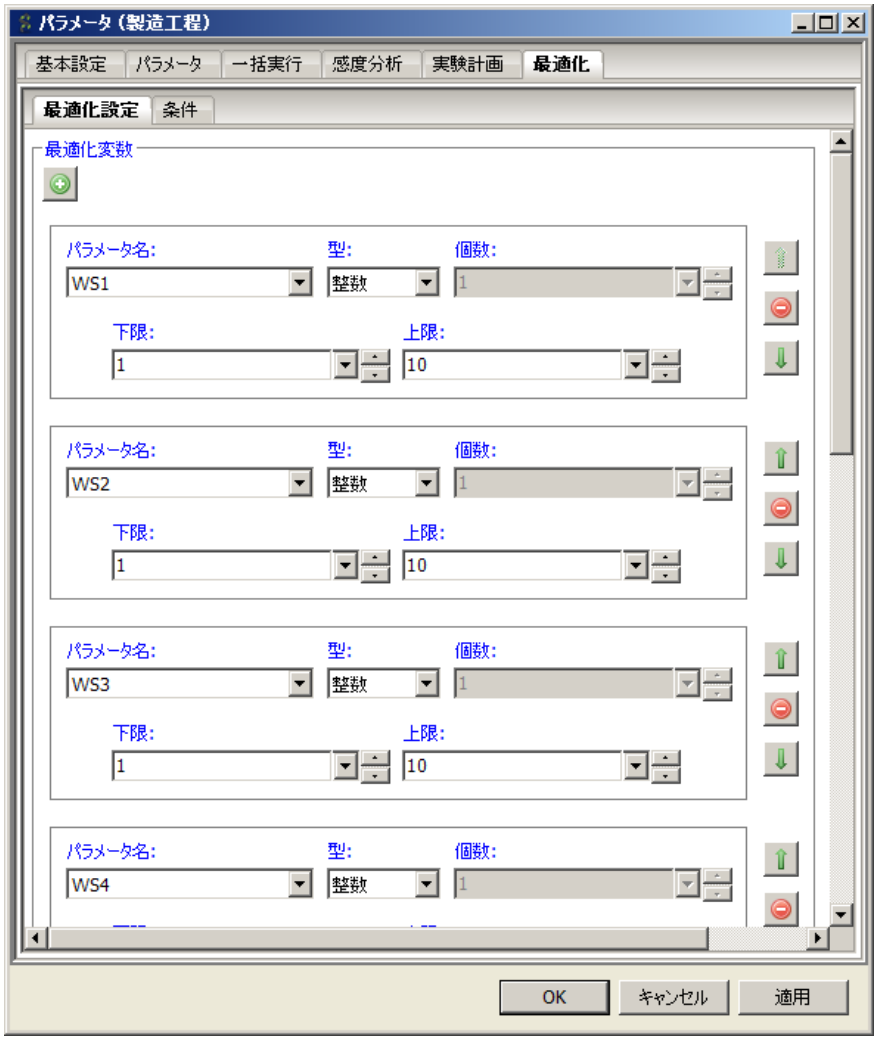


図 65: 最適化パラメータ 編集画面

図 66: 最適化パラメータ 編集画面 2

あるいは、Particle Swarm Optimizaton(PSO) と呼ばれる手法を用いて最適化を行います。最適化変数がすべて整数の場合にはランダム・サーチを行います。最適化オプションではこれらのパラメータを設定します。今回の Tutorial ではデフォルトのまま行います。この状態で「適用」ボタンを押し設定を反映させます。

次に「条件」タブをクリックし、最適化のための制約条件を指定します。ウィンドウの上側には

```
def makeConstraints():
    WS1 = IntegerVariable(name = u'WS1', lower = 1, upper = 10)
    WS2 = IntegerVariable(name = u'WS2', lower = 1, upper = 10)
    WS3 = IntegerVariable(name = u'WS3', lower = 1, upper = 10)
    WS4 = Variable(name = u'WS4', lower = 1, upper = 10)
    Buffer1 = Variable(name = u'Buffer1', lower = 1, upper = 10)
    Buffer2 = Variable(name = u'Buffer2', lower = 1, upper = 10)
    Buffer3 = Variable(name = u'Buffer3', lower = 1, upper = 10)
```

と表示されています。これは、最適化パラメータで指定した内容になります。ウィンドウの下側でこれらの変数を用いて制約条件を記述します。ここでは以下のように指定します。

```
1<=WS1+WS2+WS3+WS4<=10
```

5.6　最適化実行

「モデル」メニューから「モデルを開始する」を選択し、シミュレーション最適化を開始します。最適化を実行すると以下の画面が表示されます。上から、最適化反復数、最適化経過時間、レプリケーション番号、シミュレーション時間となります。(図 67)

最適化反復数は最適化のための反復数を表します。最適化経過時間は最適化実行の経過時間を表します。レプリケーション番号は一つのパラメータセットに対する目的関数の評価のためのレプリケーションの番号を表します。シミュレーション時間は一つのレプリケーションにおけるシミュレーション時間を表します。

グラフ表示ボタンを押すと、最適化の実行結果、それぞれの変数の値がリアルタイムに表示されます。(図 68)

結果は 5000 万円以上となり、初期の設定より最適化されていることがわかります。

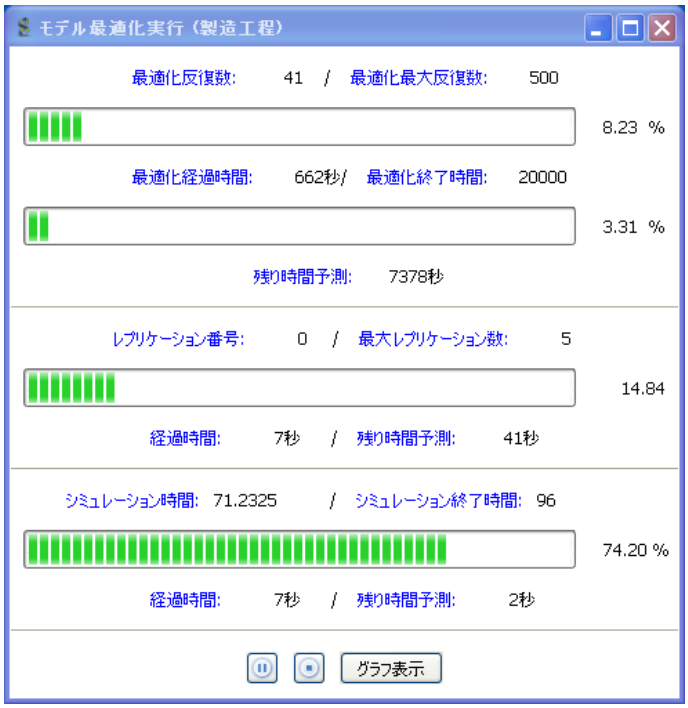


図 67: 最適化実行ダイアログ

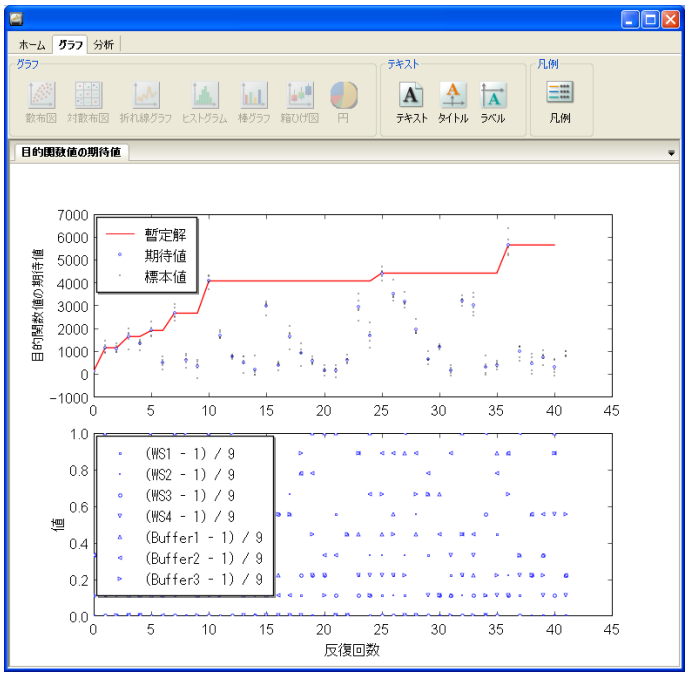


図 68: 最適化実行 グラフ表示

6 Lotka-Volterra 方程式

連続変数、連続部品を用いて「Lotka-Volterra 方程式」を解いてみます。

6.1 Lotka-Volterra モデル

Lotoka-Volterra 方程式（ロトカ-ヴォルテラ方程式）とは、捕食者と被食者の増減関係をモデル化した以下のような非線形微分方程式です。

$$\begin{aligned} \frac{dx}{dt} &= x(\alpha - \beta y) \\ \frac{dy}{dt} &= -y(\gamma - \delta x) \end{aligned}$$

ここで、x は被食者の数、y は捕食者の数、t は時間を表しています。第 1 式は、被食者の増減が、被食者の数に比例して増加し（第 1 項）、捕食者に捕食されることで減少する（第 2 項）ことを表しています。第 2 式は、捕食者の増減が、捕食者の数に比例して減少し（第 1 項）、被食者を捕食することで増加する（第 2 項）ことを表しています。

6.2 プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「Lotka-Volterra」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。次に、Lotka-Volterra モデルを作成していきます。

6.3 フローアイテムの配置

Lotka-Volterra モデルでは、微分方程式の開始のトリガを表します。ブラウザパネルで「アイテムタブ」を選択してください。アイテムタブの「アイテム」をドラッグし、モデル編集画面にドロップしてください。アイテムが配置されます。

6.4 生成部品の配置

Lotka-Volterra モデルでは、微分方程式の開始のトリガを表します。ブラウザパネルの「部品」タブを選択し、生成部品をモデル編集画面に配置します。

6.5　連続変数の配置

Lotka-Volterra 方程式において、捕食者と被食者を表します。ブラウザパネルで「変数」タブを選択し、変数タブから「連続変数」をモデル編集画面に配置します。連続変数は 2 種類あります。そこで、連続変数を 2 つ、モデル編集画面に配置します。この連続変数は Lotka-Volterra 方程式での x と y を表すので、それぞれ「x」、「y」と名前を変更します。

6.6　連続部品の配置

連続変数を用いた微分方程式を「連続」部品で定義します。ブラウザパネルで「部品」タブを選択し、「連続」部品をモデル編集画面に配置します。Lotka-Volterra 方程式を定義するので「Lotka-Volterra」と名前を変更します。

6.7　終端部品の配置

ブラウザパネルで「部品」タブを選択し、「終端」部品をモデル編集画面に配置します。このモデルにおいて、アイテムはトリガとしてしか働かないために、終端部品は必要ありませんが、出力がつながらないとエラーになるために配置します。

6.8　リアルタイムグラフ部品の配置

ブラウザパネルで「グラフ」タブを選択し、「リアルタイムグラフ」部品をモデル編集画面に配置します。このリアルタイムグラフでは、シミュレーション中の連続変数の変化の様子を表示します。

6.9　パラメータの設定

Lotka-Volterra モデルでは、4 つのパラメータ (α, β, γ, δ) を用います。それらを設定します。モデルメニューから「パラメータを編集する」を選ぶと、パラメータ編集画面が表示されます。パラメータタブを開き、図 69 のように、値を設定します。パラメータの「+」をクリックするとパラメータの編集欄が表示されます。

図 69: パラメータ編集画面

6.10　リンクの作成

Lotka-Volterra モデルにおける流れをリンクで表現します。「生成」部品と「Lotka-Volterra」部品にリンクを作成します。「Lotka-Volterra」部品と「終端」部品にもリンクを作成します。

出来上がったプロジェクトは以下のようになります。(図 70)

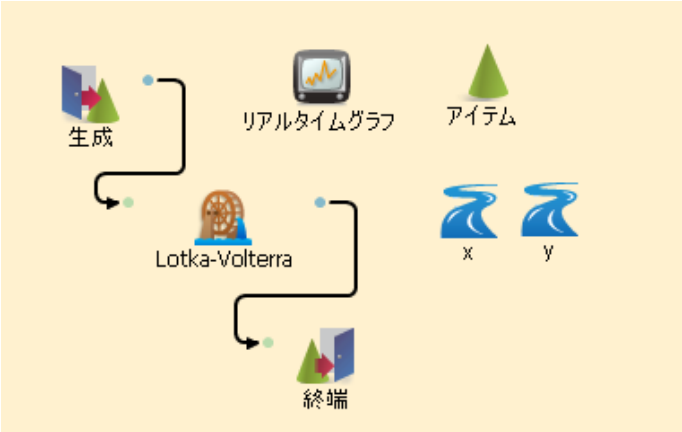


図 70: Lotka-Volterra モデル編集画面

6.11　生成部品の編集

生成されたアイテムが、連続部品に到着するとそれがトリガとなって、連続部品が動作します。今回は連続部品は 1 度だけ動作すればいいので、生成されるアイテム数は一つとなるように最大生成数を 1 としておきます。

6.12　Lotka-Volterra の編集

Lotka-Volterra 方程式を定義するために、「Lotka-Volterra」を編集します。「Lotka-Volterra」をダブルクリックすると部品編集画面が表示されます。「Lotka-Volterra」部品では、Lotka-Volterra 方程式を設定します。式の「+」をクリックし、式を追加し、図 71 のように指定します。

また、出力先として、「それ以外の出力先」として「output1(出力 1)」を選択します。

6.13　連続変数の編集

連続変数「x」、「y」を編集します。「連続変数」をダブルクリックすると編集画面が表示されます。この Lotka-Volterra モデルでは、それぞれの連続変

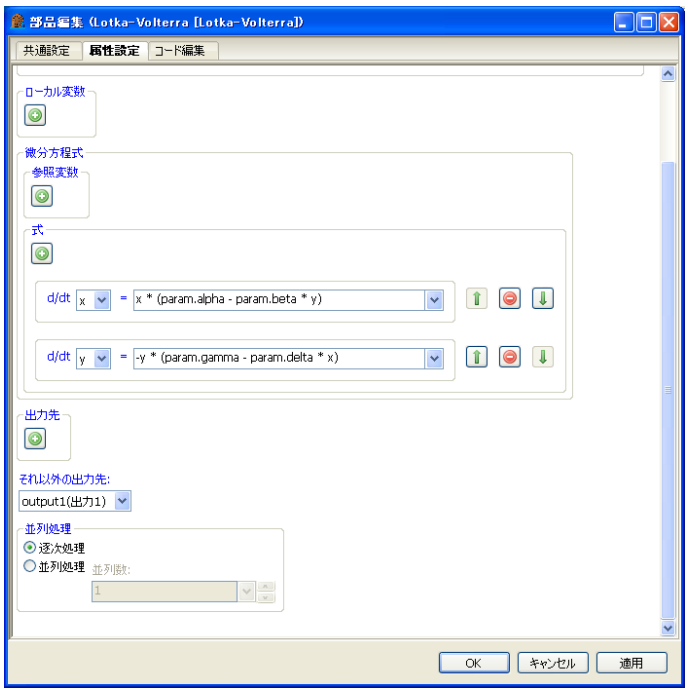


図 71: 連続部品 編集画面

数の初期値が「x」が 10000、「y」が 500 です。(図 72, 図 73)

6.14　リアルタイムグラフ部品の編集

リアルタイムグラフでは、設定時に結果が必要になるために、シミュレーションを 1 回実行したのちに設定します。以下の設定では、シミュレーションの実行が行われ結果がある状態とします。

「リアルタイムグラフ」をダブルクリックすると編集画面が表示されます。「グラフ」タブで「折れ線グラフ」をクリックします。すると、折れ線グラフ設定ウインドウが表示れれるので、グラフデータの x と y を設定します。x のセルをクリックすると設定可能なデータが表示されるので、そこから x(出力) の「時間」を設定します。対応する y のセルには、x(出力) の「値」を設定します。y も表示するので、同様に、別な行で、x のセルをクリックして、y(出力) の「時間」を設定し、対応する y のセルをクリックし、y(出力) の「値」を設定します。(図 74)

設定すると、以下のようなグラフが作成されます。(図 75)

この状態で実行すると、リアルタイムに変数の変化の様子が表示されます。

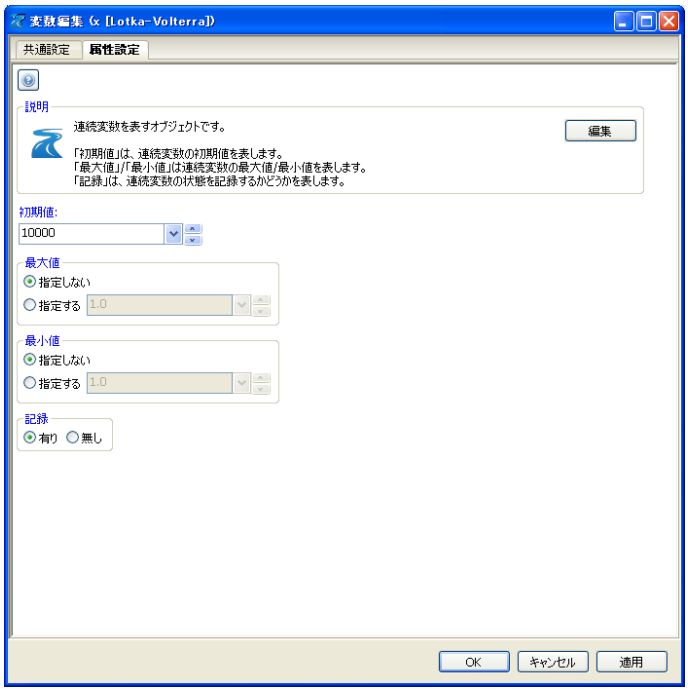


図 72: 連続変数 x 編集画面

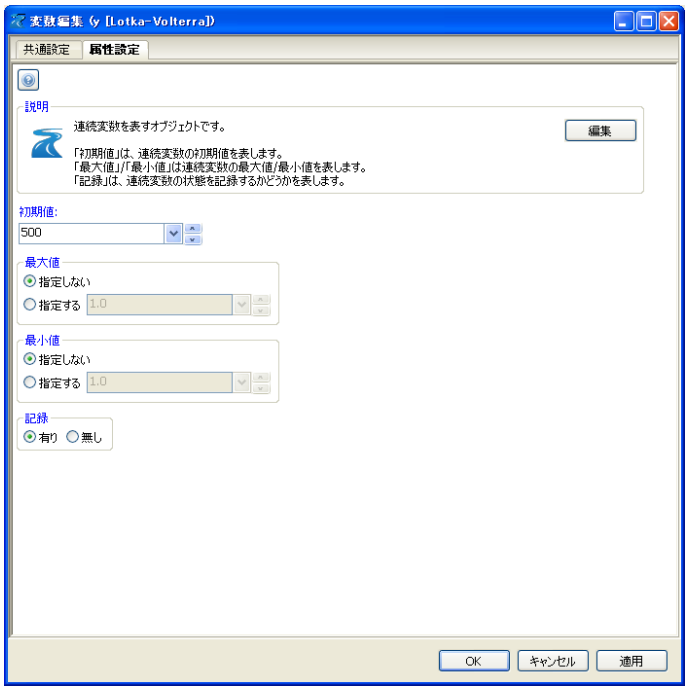


図 73: 連続変数 y 編集画面

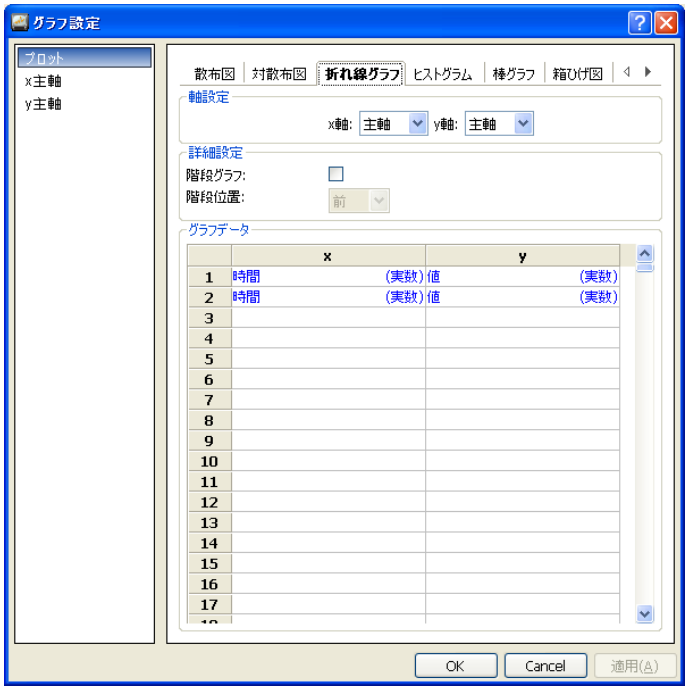


図 74: リアルタイムグラフ 編集画面

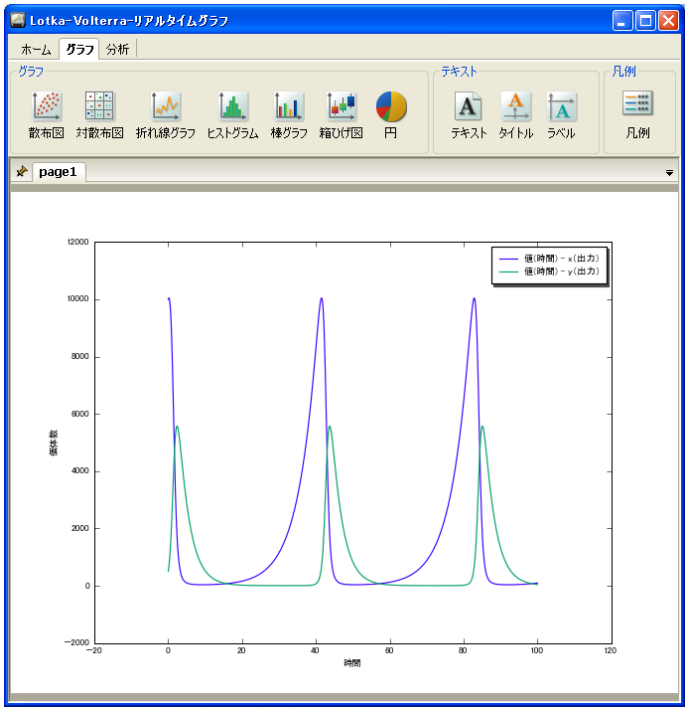


図 75: リアルタイムグラフ

7　同期エージェント

同期エージェントは、エージェントの一種ですが、特に全てのエージェントの状態が固定タイミングで変化するようなエージェントの事です。ここではライフゲームを例に説明します。

7.1　ライフゲーム

ライフゲームはセル・オートマトンの一種です。

シミュレーション空間上に多数のセル（細胞）が配置されていて、それぞれのセルには近傍があるとします。各セルは状態を持っていますが、各セルの次の状態は近傍のセルの状態によって決定するとしたモデルです。

ライフゲームは、以下のような単純なルールの基でシミュレーションを行う事で複雑な結果が得られるセルオートマトンの 1 例です。

ライフゲームは以下のようなルールでシミュレーションを行います。

各セルは状態を持っており、各セルは以下のルールで「生」と「死」の状態を遷移します。

- 自分の状態が「生」の場合
  - 回りに生きたセルが 1 以下の場合、死滅する。(過疎)
  - 回りに生きたセルが 2 または 3 の場合、生存する。
  - 回りに生きたセルが 4 以上の場合、死滅する。(過密)
- 自分の状態が「死」の場合
  - 回りに生きたセルが、3 の場合、誕生する。
  - それ以外の場合、変化しない。

7.2　エージェントモデリング

ここで、ライフゲームをどのようにエージェントモデル化するかを考えます。

ライフゲームのシミュレーション空間は、セルで構成されています。各セルは、状態を持っており、また、各セルは、隣接するセルがあります。

ここで「セルが状態を持つ」と考えるよりは、各セルに「状態を持ったエージェントが配置される」と考えると、エージェントとしてモデリングがしやすくなります。

まず環境としては S4 Simulation System で用意されている 8 方格子がそのまま使えます。

各エージェントはセル上に、つまり 8 方格子のノード上に 1 人ずつ配置するようにします。その上で、各エージェントは上記のルールで「生」と「死」の状態を遷移するようにします。

エージェントの持つ変数は以下になります。

- screenColor: そのセルの表示色
- screenSize: そのセルの表示サイズ
- state: 状態 (”living” / ”dead”)

エージェント集合の持つ変数は以下になります。

- gpos: 全ノードを列挙するイテレータ

7.3　プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「ライフゲーム」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。次に、ライフゲームモデルを作成していきます。

7.4　環境の作成

ブラウザパネルで「環境タブ」を選択してください。環境タブの「格子グラフ」をドラッグし、モデル編集画面にドロップしてください。格子グラフが配置されます。

「格子グラフ」をダブルクリックすると編集画面が表示されます。(図 76)

ここでは、特に変更する必要はありません。

7.5　同期エージェントの作成

ブラウザパネルで「エージェントタブ」を選択してください。エージェントタブの「同期エージェント」をドラッグし、モデル編集画面にドロップしてください。同期エージェントが配置されます。

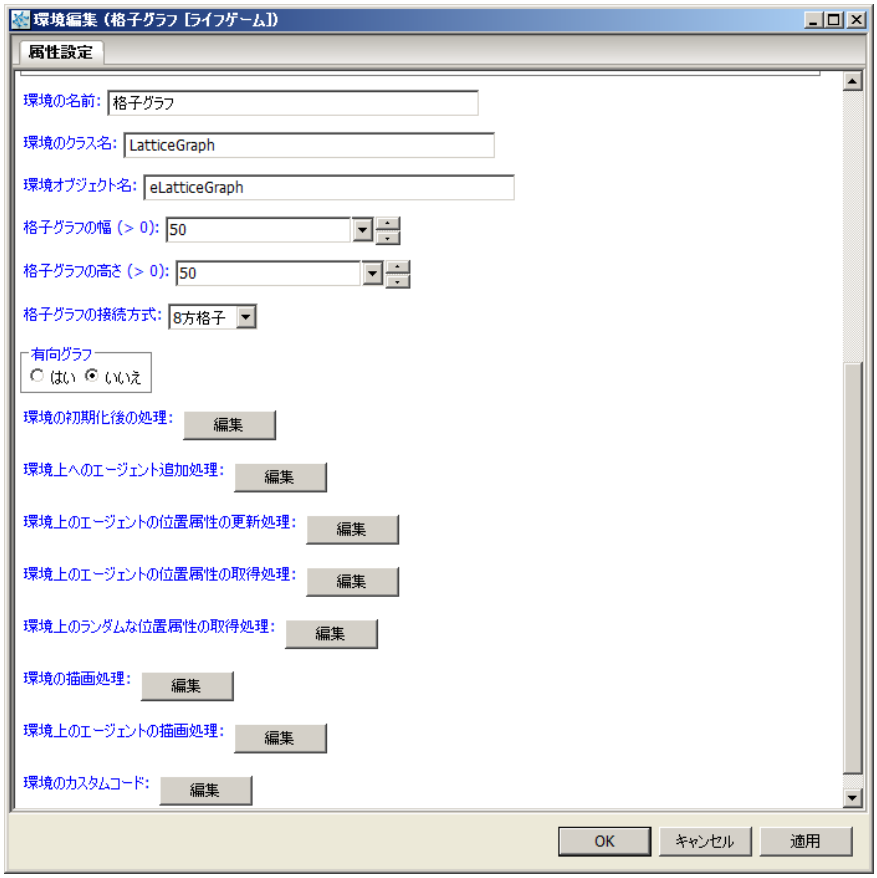


図 76: 格子グラフ編集画面

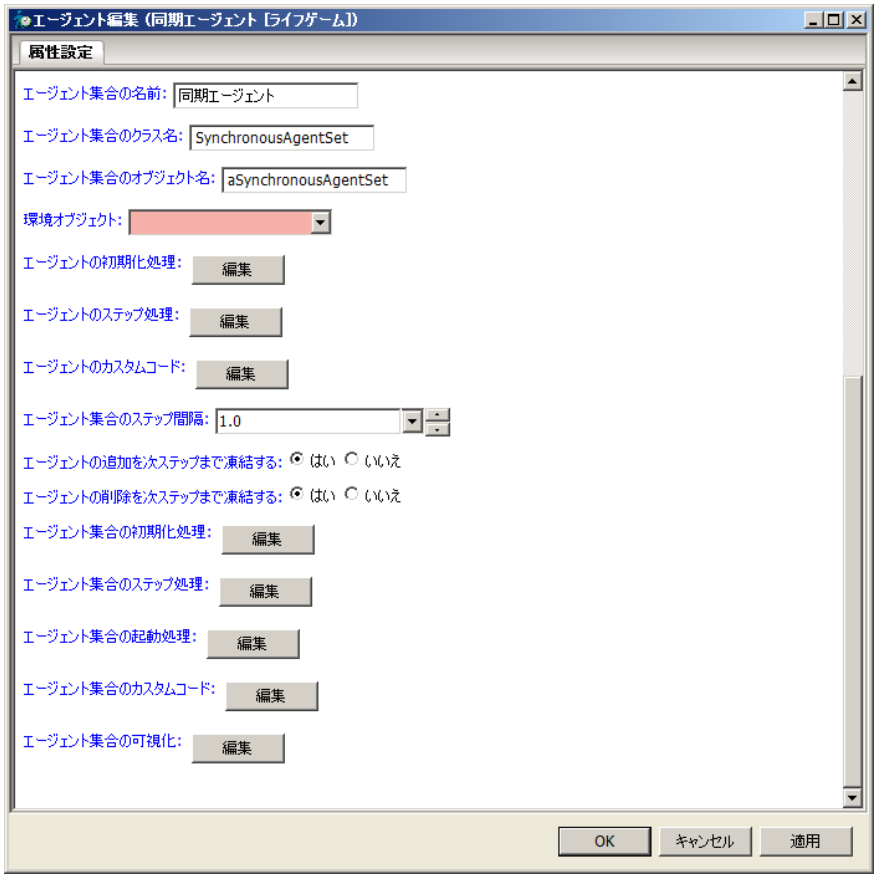


図 77: 同期エージェント編集画面

7.6　エージェントの環境の設定

「同期エージェント」をダブルクリックすると編集画面が表示されます。(図 77)

まず、エージェントと環境を結び付けます。「環境オブジェクト」のプルダウンメニューで、「eLatticeGraph(格子グラフ)」を選択して下さい。

7.7　エージェントの初期化処理の編集

「同期エージェント」の属性設定タブにて、「エージェントの初期化処理」の「編集」をクリックします。すると、エージェントの初期化処理という名前のタブが開かれます。そのタブ内で、以下のようなコードを入力します。(図 78)

エージェントの初期化処理

```
self.screenColor = 'b'
self.screenSize = 30
self.setPosition(next(self.agentset.gpos))
self.state = next(sample(["living", "dead"]))
```

「エージェントの初期化処理」では、個々のエージェントの初期化を行います。

各エージェントの表示色、表示サイズを設定しています。しかし、この値は仮のものなので、あまり意味はありません。

次に、各エージェントの初期位置を設定しています。ここは慣れていないと少々難しいかもしれません。self.agentset.gpos は、後の「エージェント集合の初期化」で設定しますが、全ノードを列挙するイテレータです。つまり、next(self.agentset.gpos) を呼ぶ度に、ノードを次々と返してくれます。結果として、それぞれのエージェントが、別々のノードに配置されます。エージェント数がちょうどノード数と一致するなら、全てのノードに 1 エージェントずつ割り振られた状態になります。

最後に、各エージェントの状態 (生/死) をランダムに定めています。

7.8　エージェントのステップ処理の編集

同期エージェントの「属性設定」タブに戻り、「エージェントのステップ処理」の「編集」をクリックし、以下のようなコードを入力します。(図 79)

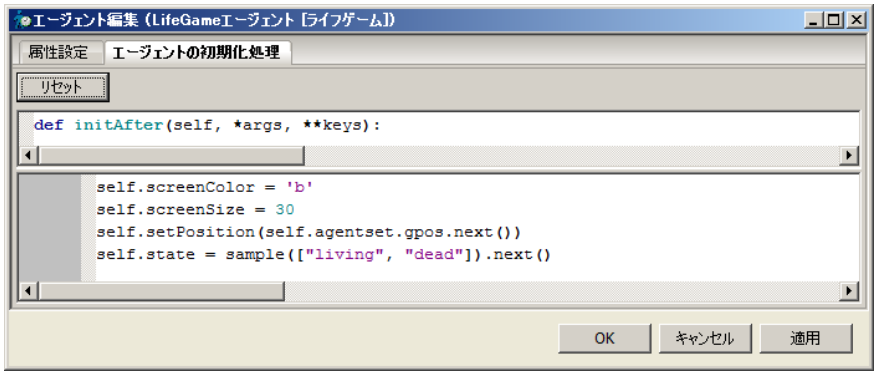


図 78: 同期エージェント編集画面 (エージェントの初期化処理)

エージェントのステップ処理

```
# 1 近傍のエージェント群 (自分以外)
agents = self.findNeighborAgents(d = 1)
# 近傍の生存数
nliving = len([a for a in agents
               if a.state == "living"])
# 自分と近傍の生存数による場合わけ
if self.state == "living":
    if nliving in (2, 3):
        self.state = "living" # 生存
    else:
        self.state = "dead" # 過疎/過密
else:
    if nliving == 3:
        self.state = "living" # 誕生
    else:
        self.state = "dead"
# 状態により、表示サイズを変更
if self.state == "living":
    self.screenSize = 30
else:
    self.screenSize = 0
```

「エージェントのステップ処理」では、個々のエージェントのステップ処理を行います。ここがシミュレーションの本体になります。

「self.findNeighborAgents(d = 1)」は 1 近傍のエージェント群 (自分以外) をリスト形式で返却します。格子グラフの形状から、この場合の 1 近傍は、縦横と斜めに隣接した点を含みます。

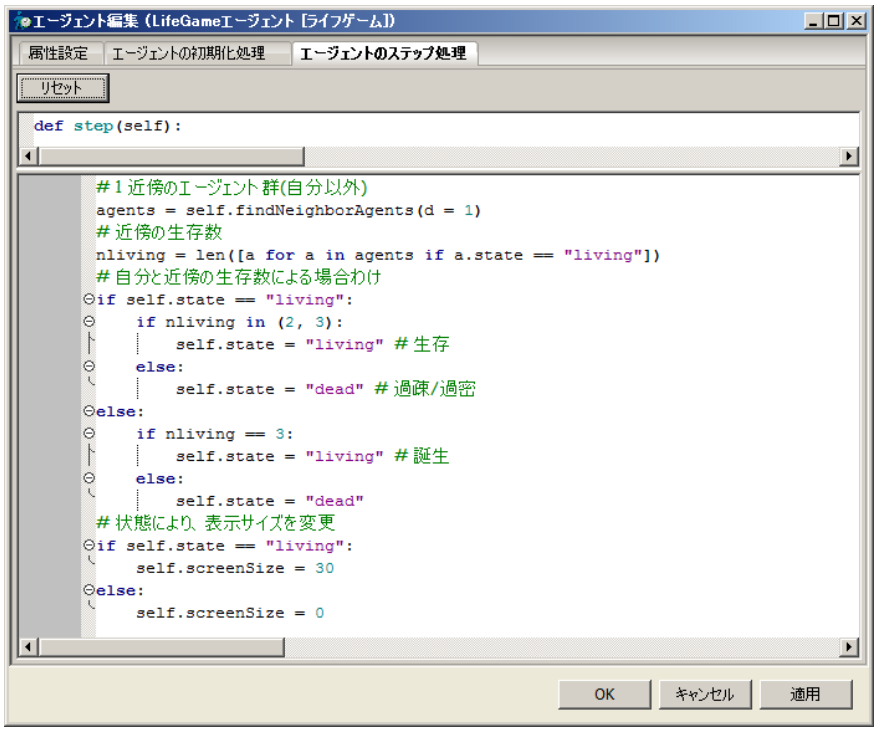


図 79: 同期エージェント編集画面 (エージェントのステップ処理)

つまり、セルの構造などは特に意識しなくとも、自分の回りいるエージェントを列挙できます。

「len([a for a in agents if a.state == ”living”])」は、自分の回りにいるエージェントの中で生存状態であるエージェントの数を数えあげています。

その後は、上記のライフゲームのルールに従い、自分の状態と回りの生きたセルの個数によって、自分の状態を更新しています。

最後に、求まった状態によって表示上のサイズを変更します。

ここで重要な事があります。これらのエージェントのステップ処理は、個々のエージェントで実行されますが、どのような順番で呼び出されるかは明らかではありません。

あるエージェントの状態を変更してしまうと、別のエージェントから見るとそのエージェントの回りの生きたセルの個数が変わってしまう事になります。

この問題に対処するには、今の状態と次の世代の状態を持たせ、最初に全てのエージェントの次の世代の状態を計算し、その後、全てのエージェントの状態に反映させるという方法もあります。もしそのような事を行いたいのなら、「エージェント集合のステップ処理」を修正する事で対応可能です。

しかし、このように同一世代で属性の変更を遅延させたい場合は、「凍結変数」と呼ばれる仕組みを利用する事で簡単に対処出来ます。

現在作成しているモデルでは、state を凍結変数にしています。つまり、状態 state は、値を更新しても、同一世代内では、別のエージェントから参照しても古い値のままになっています。つまり、単純にルール通りに実装した上記のようなコードで、矛盾なく動作します。

7.9 エージェント集合の初期化処理の編集

同期エージェントの「属性設定」タブに戻り、「エージェント集合の初期化処理」の「編集」をクリックし、以下のようなコードを入力します。

エージェント集合の初期化処理

```
self.agentFreezeVars = ["state"] # エージェントの凍結する変数リスト
self.gpos = iter(self.env.nodes())
# エージェントの生成
self.generateAgents(len(self.env.nodes()))
```

「エージェント集合の初期化処理」では、エージェント集合の初期化を行います。

最初に、エージェントの凍結する変数リストを宣言しています。先程の説明の通り、ここでは、state を凍結変数にしています。

次に、各エージェントの初期化処理で使う、gpos を宣言しています。全ノードを列挙するイテレータを設定しています。

最後にエージェントを生成しています。ここでは、全ノードにちょうど 1 エージェントずつ割り当てたいので、ちょうどノード数分のエージェントを作成しています。

7.10　エージェント集合の可視化の編集

同期エージェントの「属性設定」タブに戻り、「エージェント集合の可視化」の「編集」をクリックし、以下のようなコードを入力します。(図 80)

エージェント集合の可視化

```
interval = 1 # 表示間隔
screen = self.getAgentScreen(interval = interval,
                             xlim = (0,1), ylim = (0,1))
screen.addAgentSet(self)
screen.start()
```

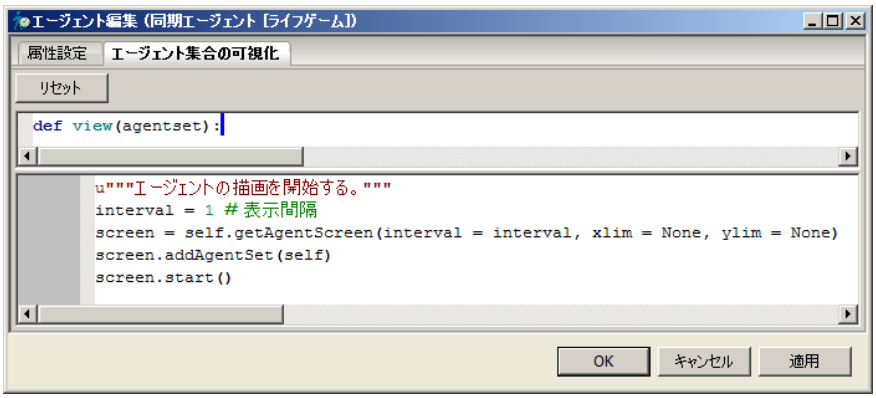


図 80: 同期エージェント編集画面 (エージェント集合の可視化)

「エージェント集合の可視化」ではエージェント集合の可視化コードを記述します。

実際の描画本体は、格子グラフの「環境の描画処理」、「環境上のエージェントの描画処理」に記述されています。カスタマイズしたい場合は、こちらを編集する事で、調整できます。通常の動作では、各エージェントに設定された属性 screenColor, screenSize に従って各エージェントが描画されます。

7.11　モデルの実行

モデルメニューから「モデルを開始する」を選ぶと、モデルが実行されます。

以下のようなエージェントフレームが表示されます。(図 81)

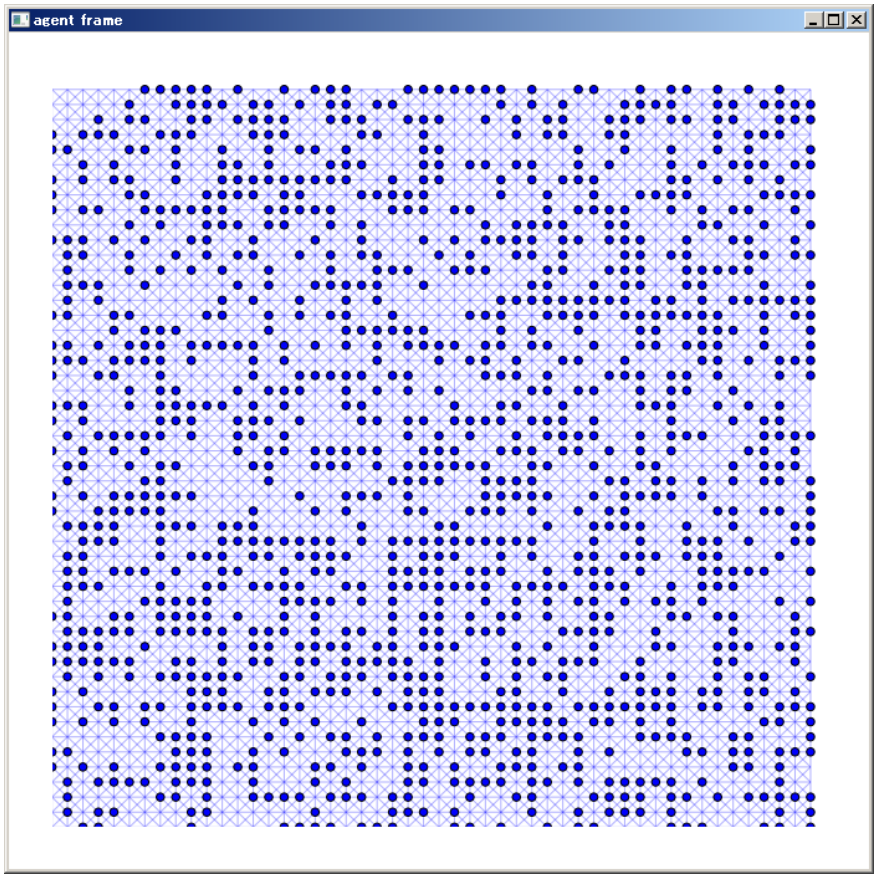


図 81: ライフゲーム実行結果 (初期配置)

時間経過と共に状態が変化していきます。100 期後の状態は以下のようになります。(図 82)

8　非同期エージェント

非同期エージェントは、エージェントの一種ですが、特に個々のエージェントの状態が独立に離散タイミングで変化するようなエージェントです。ここでは、ツイッターの伝播シミュレーションを例に説明します。

同期エージェントは、全てのエージェントの状態が固定タイミングで変化しますが、非同期エージェントはそうではなく、他エージェントからのイベントを契機に動いたり、エージェントごとにランダムな時間スリープし、その後に何らかの動作を行ったりします。

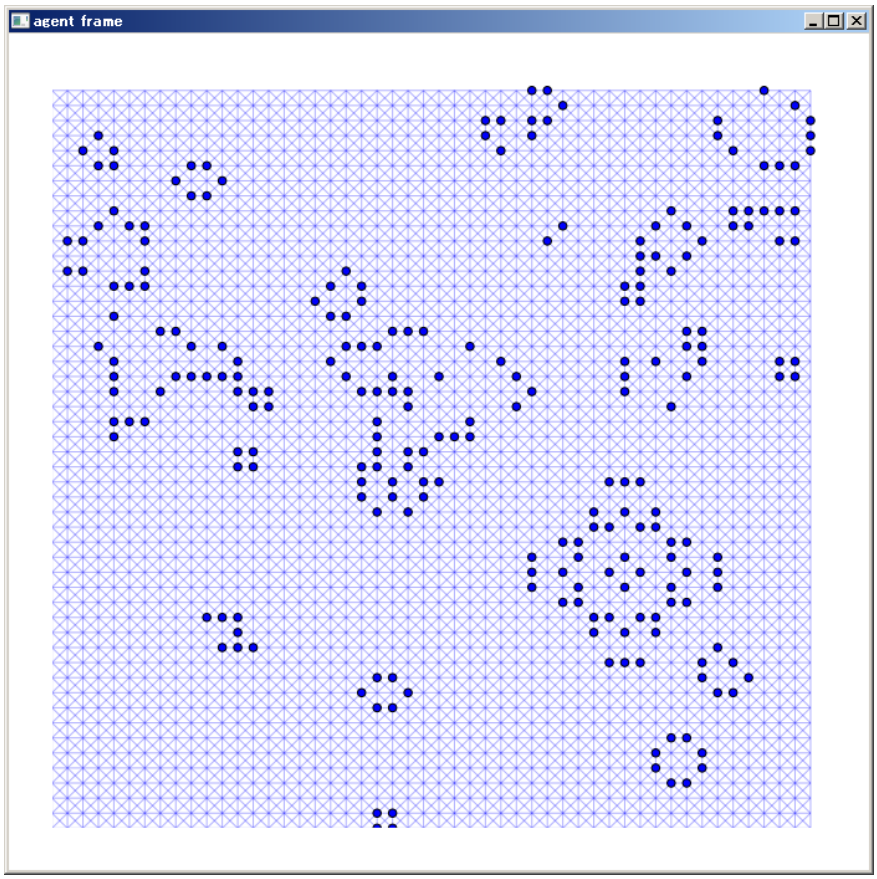


図 82: ライフゲーム実行結果 (100 期後)

8.1　ツイッターの伝播シミュレーション

ツイッターの伝播シミュレーションは、ある CM を視聴したユーザーの発したツイートがどのように伝播するかをシミュレーションします。

ある CM を視聴したユーザーは一定の確率で、その CM に関する内容をツイートします。さらに、そのツイートを見たユーザーは、一定の確率である時間後にリツイートを行います。そのような想定で、そのツイートを見た人数がどのように変化するのかのシミュレーションを行います。

8.2　エージェントモデリング

ここで、ツイッターの伝播シミュレーションをどのようにエージェントモデル化するかを考えます。

同期エージェントのサンプルで示したライフゲームと同様に、グラフ上の各ノードに、1 人ずつ配置します。リンクはツイッターの接続関係を示します。ここでは、必ず双方向のリンクがあるとします。

環境としては、スケールフリーネットワーク (後述) を利用します。

エージェントの持つ変数は以下になります。

- screenColor: そのエージェントの表示色
- screenSize: そのエージェントの表示サイズ
- state: 状態 (”init” / ”imp” / ”tweet”)
- wp: ある CM を視聴する確率
- tp: ある CM を視聴した時ツイートする確率
- tt: ある CM を視聴した時にツイートするまでの時間 (指数分布)
- rtp: ある CM に関する CM のツイートを見た時に、リツイートする確率
- rtt: ある CM に関する CM のツイートを見た時に、リツイートする時間 (指数分布)
- spool: ツイートを受信するスプール

同期エージェントと違って、各エージェントによって、ツイートするまでの待ち時間やリツイートするまでの待ち時間が異ります。また、リツイートの処理が走るタイミングはツイートを見たタイミングのみとなります。このような場合は、非同期エージェントとしてモデリングを行います。

エージェント集合の持つ変数は以下になります。

- gpos: 全ノードを列挙するイテレータ
- gp: 0 から 1 の一様分布乱数生成器
- nimp: ツイートを見た総人数
- monitor: ツイートを見た総人数の時系列変化を記録する時系列モニター

8.3　プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「ツイッターの伝播シミュレーション」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。次に、ツイッターの伝播シミュレーションモデルを作成していきます。

8.4　環境の作成

ブラウザパネルで「環境タブ」を選択してください。環境タブの「BarabasiAlbert グラフ」をドラッグし、モデル編集画面にドロップしてください。BarabasiAlbert グラフが配置されます。

BarabasiAlbert グラフとは、スケールフリーネットワークを作成します。

スケールフリーネットワークとは、現実世界に存在するネットワークの性質を模したネットワーク構造です。

例えば、ウェブページをノードとした場合、ノード同士がハイパーリンクでリンクされています。ごく少数のハブと呼ばれる有名サイトは大量のサイトから参照されますが、大多数のサイトは少ないサイトからしか参照されません。このような特徴は、スケールフリー性 (次数分布のべき乗則) と呼ばれます。

ツイッターの参照関係もスケールフリー性を持っている事から、BarabasiAlbert グラフを利用します。

「BarabasiAlbert グラフ」をダブルクリックすると編集画面が表示されますので、ノード数に「1000」を入力して下さい。

その他は変更する必要はありません。

8.5　非同期エージェントの作成

ブラウザパネルで「エージェントタブ」を選択してください。エージェントタブの「非同期エージェント」をドラッグし、モデル編集画面にドロップ

してください。非同期エージェントが配置されます。

8.6　エージェントの環境の設定

「非同期エージェント」をダブルクリックすると編集画面が表示されます。

まず、エージェントと環境を結び付けます。「環境オブジェクト」のプルダウンメニューで、「eBarabasiAlbertRandomGraph(BarabasiAlbert グラフ)」を選択して下さい。

8.7　エージェントの初期化処理の編集

「非同期エージェント」の属性設定タブにて、「エージェントの初期化処理」の「編集」をクリックします。すると、エージェントの初期化処理という名前のタブが開かれます。そのタブ内で、以下のようなコードを入力します。

エージェントの初期化処理

```
self.setPosition(next(self.agentset.gpos))
# ある CM を視聴する確率
self.wp = 0.2
# ある CM を視聴した時ツイートする確率
self.tp = 0.15
# ある CM を視聴した時にツイートするまでの時間 (指数分布)
self.tt = exponentialDistribution(10 * 60)
# ある CM に関する CM のツイートを見た時に、リツイートする確率
self.rtp = 0.1
# ある CM に関する CM のツイートを見た時に、リツイートする時間 (指
数分布)
self.rtt = exponentialDistribution(10 * 60)
# 状態
#  - init: 初期状態
#  - imp: ツイートを一度でも目にした
#  - tweet: ツイートもしくはリツイートした
self.state = "init"
self.spool = Store()
```

各エージェントは、それぞれ別々のノードに、割り振られます。

エージェントの動作を定める各定数を設定しています。視聴確率などは、定数を設定しています。待ち時間については、指数分布に従う乱数系列を設定しています。

状態は、3 状態あります。初期状態は、”init” です。

- init: 初期状態
- imp: ツイートを一度でも目にした
- tweet: ツイートもしくはリツイートした

spool は、ツイートを受信するスプールです。離散イベントシミュレーションの資源 Store として表現しています。

8.8　エージェントのプロセス処理の編集

非同期エージェントの「属性設定」タブに戻り、「エージェントのプロセス処理」の「編集」をクリックし、以下のようなコードを入力します。

注意: エージェントのステップ処理 (1) とエージェントのステップ処理 (2) に分かれていますが、両方を「エージェントのプロセス処理」内につなげて入力します。

エージェントのステップ処理 (1)

```
# 状態を init にする
self.state = "init"
self.screenSize = 0
self.screenColor = "w"

# ある CM を視聴するか
if next(self.agentset.gp) < self.wp:
    # ある CM を視聴した時ツイートするか
    if next(self.agentset.gp) < self.tp:
        # ある CM を視聴した時にツイートするまでの時間
        yield pause(next(self.tt))
        # フォロワーにツイート
        agents = self.findNeighborAgents(d = 1)
        for agent in agents:
            yield agent.spool.put1("MSG")
        # 状態を tweet にする
        self.state = "tweet"
        self.screenSize = 30
        self.screenColor = "r"
        self.agentset.nimp += 1
        self.agentset.monitor.observe(
            now(),
            self.agentset.nimp)
```

エージェントのステップ処理 (2)

```
# リツイートプロセス
# 一度でもツイートしたら完了
while self.state != "tweet":
    # メッセージを受けとるまで待ち受ける
    result = yield self.spool.get1(name = "tweet")
    tweet = result["tweet"]
    # 状態を imp にする
    self.state = "imp"
    self.screenSize = 30
    self.screenColor = "g"
    self.agentset.nimp += 1
    self.agentset.monitor.observe(now(), self.agentset.nimp)
    # ある CM に関する CM のツイートを見た時に、リツイートするか
    if next(self.agentset.gp) < self.rtp:
        # ある CM に関する CM のツイートを見た時に、リツイートす
る時間
        yield pause(next(self.rtt))
        # フォロワーにリツイート
        agents = self.findNeighborAgents(d = 1)
        for agent in agents:
            yield agent.spool.put1(tweet)
        # 状態を tweet にする
        self.state = "tweet"
        self.screenSize = 30
        self.screenColor = "r"

yield alwaysFalse()
```

大きくわけて、2 つの処理に分かれます。

最初の処理は、ある CM を視聴した時の処理です。

各エージェントは最初に一定の確率で CM を視聴します。視聴した場合、一定の確率で、指数分布に従った時間後にフォロワーにツイートを行います。近傍をみつけるには、findNeighborAgnets を使っています。

各エージェントは、ツイートを受信するスプールを持っています。エージェント A がエージェント B にツイートする場合は、B の spool にツイートを追加するという表現しています。今回スプールに蓄えられるツイート数に制限はないので、スプールへの追加処理は即時に成立しますが、Store への追加処理は必ず、

```
yield agent.spool.put1("MSG")
```

のように記述する必要があります。詳しくは、資源 Store のマニュアルをご参照下さい。

ツイートした場合は、状態が変わりますので、state, screenSize, screenColor を変更します。

シミュレーション全体で、ツイートを見た人数をカウントし、エージェント集合の nimp に記録していますので、ここで更新しています。また、nimp の時系列変化も記録します。

次の処理は、リツイート処理です。

リツイート処理は常に走っているわけではありません。他エージェントからツイートを受けた時のみに起動します。

もし状態が ”tweet” なら、終了します。そうでないなら、以下を繰り返します。

まず、

```
result = yield self.spool.get1(name = "tweet")
```

にて、自分のスプール内にあるツイートを取り出します。もし、スプールが空なら、ツイートを受けとるまで、待ち受け続けます。待っている間は CPU を消費しません。この部分が離散イベント処理になります。

ツイートを受けとった場合、result[”tweet”] にて、そのツイートを受け取る事ができます。

ツイートを受けとったら、状態が変わりますので、state, screenSize, screenColor を変更します。

ツイートを見た人数が増えますので nimp, monitor を更新します。

その後、一定の確率で、指数分布に従った時間後にフォロワーにリツイートを行います。その中身の処理は初回のツイート処理と全く同様です。

8.9 エージェント集合の初期化処理の編集

同期エージェントの「属性設定」タブに戻り、「エージェント集合の初期化処理」の「編集」をクリックし、以下のようなコードを入力します。

エージェント集合の初期化処理

```
nodes = self.env.nodes()
self.gpos = iter(nodes)
self.gp = uniformDistribution(0, 1)
self.nimp = 0
self.monitor = TimeMonitor([u"視聴者数"], ["i"], name = u"
視聴者数")
self.simulator.addMonitor(self.monitor)
# エージェントの生成
self.generateAgents(len(nodes))
```

ライフゲームと同様に、各エージェントの初期化処理で使う、gpos を宣言しています。全ノードを列挙するイテレータを設定しています。

gp は、0 から 1 の一様分布乱数生成器です。

nimp はツイートを見た総人数です。ここでは 0 に初期化しています。

monitor はツイートを見た総人数の時系列変化を記録する時系列モニターです。データ列を 1 つ持つ時系列モニターで、その列の名前を「視聴者数」としています。また、モニターの名前も「視聴者数」としています。

エージェント集合内で作ったモニターは、必ず simulator に登録して下さい。登録処理は、self.simulator.addMonitor で行います。これを行わないと、結果が保存されませんのでご注意下さい。

最後にエージェントを生成しています。ここでは、全ノードにちょうど 1 エージェントずつ割り当てたいので、ちょうどノード数分のエージェントを作成しています。

8.10　エージェント集合の可視化の編集

非同期エージェントの「属性設定」タブに戻り、「エージェント集合の可視化」の「編集」をクリックし、以下のようなコードを入力します。

エージェント集合の可視化

```
interval = 60 # 表示間隔
screen = self.getAgentScreen(interval = interval,
                             xlim = None, ylim = None)
screen.addAgentSet(self)
screen.start()
```

表示の更新間隔を 60 秒 (シミュレーション時間) としています。

8.11　モデルパラメータの変更

モデルメニューから「パラメータを編集する」を選ぶと、パラメータ編集画面が表示されます。

初期設定では、シミュレーション終了時間が 「100.0」(秒) になっています。これでは十分なシミュレーションができません。

ここでは、シミュレーション終了時間を、「終了時間あり」で、「60 * 60 * 0.5」に設定します。これはシミュレーション時間を 30 分に設定しています。

8.12　モデルの実行

モデルメニューから「モデルを開始する」を選ぶと、モデルが実行されます。以下のようなエージェントフレームが表示されます。(図 83)

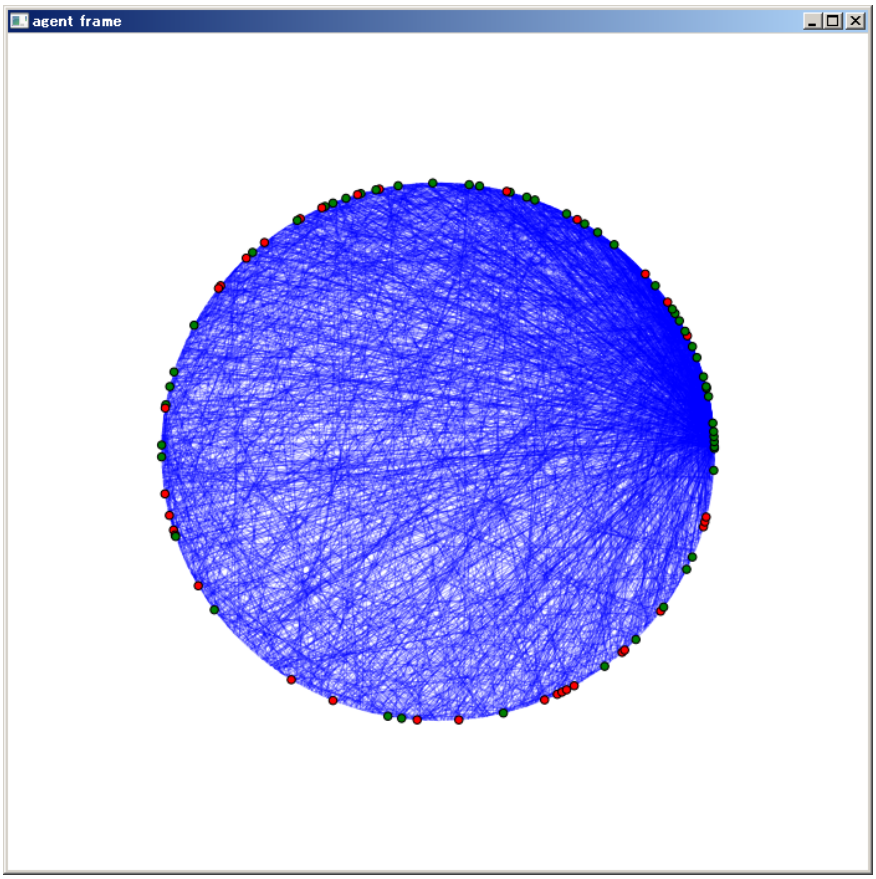


図 83: ツイートの伝播結果

青色で表示されているのは、エージェント間のリンクです。

赤色で示されているエージェントは、ツイートを行ったエージェントで、緑色は、一度でもツイートを目にしたエージェントです。

8.13　リアルタイムグラフ

ブラウザパネルで「グラフタブ」を選択してください。グラフタブの「リアルタイムグラフ」をドラッグし、モデル編集画面にドロップしてください。リアルタイムグラフが配置されます。

「リアルタイムグラフ」をダブルクリックすると編集画面が表示されます。

折れ線グラフをクリックすると、グラフ設定画面が表示されますので、x 軸に視聴者数の時間、y 軸に視聴者数の視聴者数を選択して下さい。(図 84)

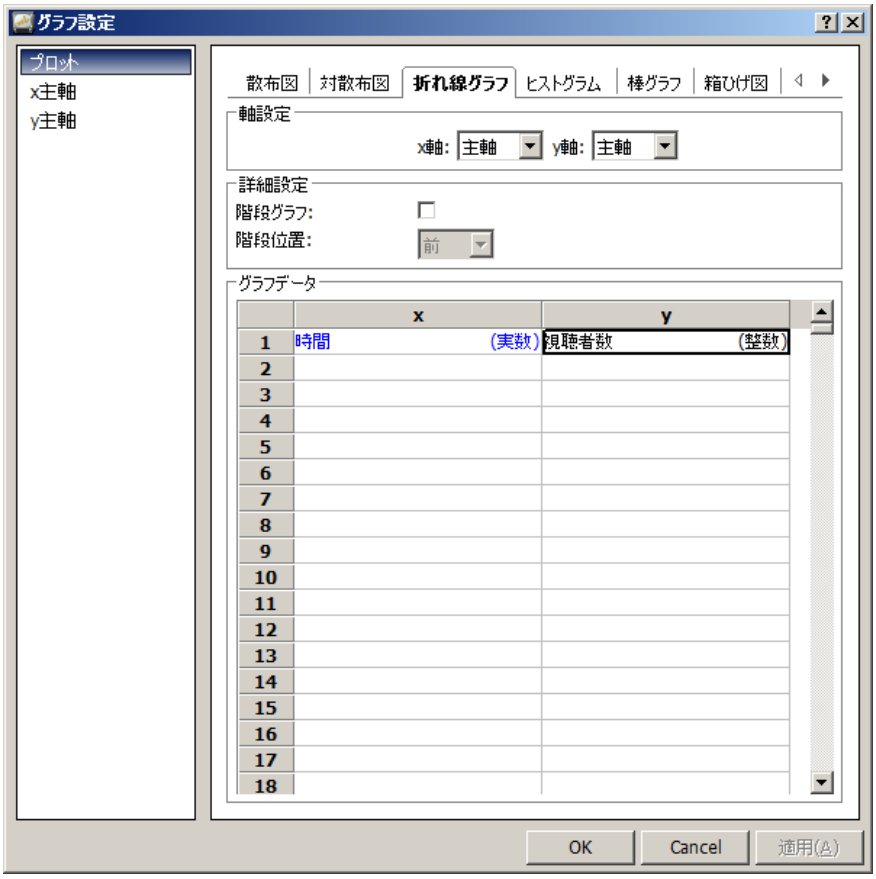


図 84: ツイートエージェントの視聴者数の時系列変化の設定

このグラフは、ツイートエージェントの視聴者数の時系列変化を示します。以下のようなグラフが表示されます。(図 85)

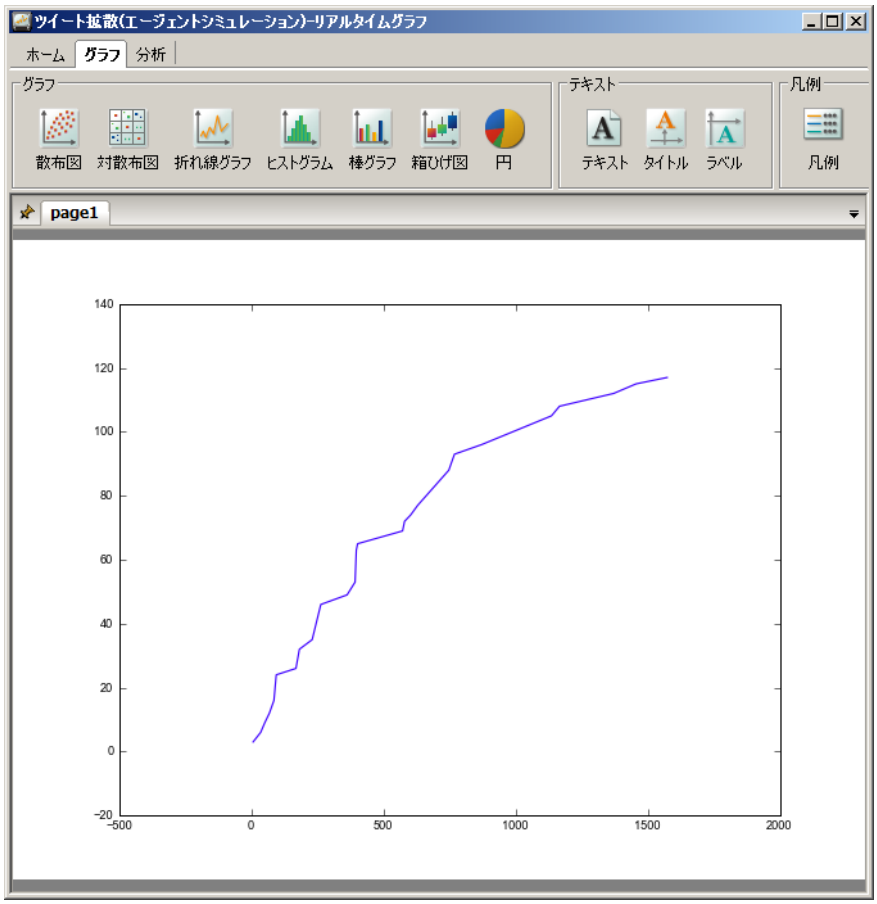


図 85: ツイートエージェントの視聴者数の時系列変化

8.14　Gephi による配置

ツイートの伝播結果を可視化しましたが、グラフが円状に表示されてしまうので、あまり見易いとは言えません。

ここでは、Gephi と呼ばれるツールを使って、グラフの自動配置を行ってみます。

- The Open Graph Viz Platform:
  - http://gephi.org/

より、Gephi 0.9.2 をインストールします。

スタートメニューより、Gephi を起動します。「ようこそ」というウィンドウが表示された場合は、ひとまず、右上の「×」をクリックし閉じます。

ここで、S4 Simulation System で作成したグラフを取り込みます。まず、BarabasiAlbert グラフをエクスポートします。

BarabasiAlbert グラフで右クリックを押すと表示されるメニューから、「グラフファイルを出力する」を選択します。

すると、「グラフファイルの出力」ウィンドウが表示されますので、GraphML フォーマットを選択します。次に、ファイルダイアログが表示されますので、適当な位置に保存して下さい。(Gephi から参照しますので、保存した場所を覚えておいて下さい)

Gephi の画面に戻り、「ファイル」メニューから「開く」を選択します。ファイルダイアログが表示されますので、先程保存した graphml 形式のファイルを選択して下さい。

以下のような画面が表示されます。(図 86)

左側の「レイアウト」から配置アルゴリズムを選択します。利用可能な配置アルゴリズムは、バージョンによって異なりますが、ここでは、「Fruchterman Reingold 」を選択し、「実行」を押します。配置アルゴリズムによって、様々なパラメータが用意されています。それらの意味は Gephi のマニュアルをご参照下さい。

自動配置が終わらない場合は、「中止」ボタンを押して下さい。

自動配置が終了すると以下のようになります。(図 87)

自動配置が終了したら、「ファイル」メニューから、「エクスポート」→「グラフファイル」を選択すると、ファイルダイアログが表示されます。まず、ファイルのタイプを「GraphML ファイル (*.graphml)」にして下さい。次に適当なフォルダ上に適当な名前で保存して下さい。(S4 Simulation System から参照しますので、保存した場所を覚えておいて下さい)

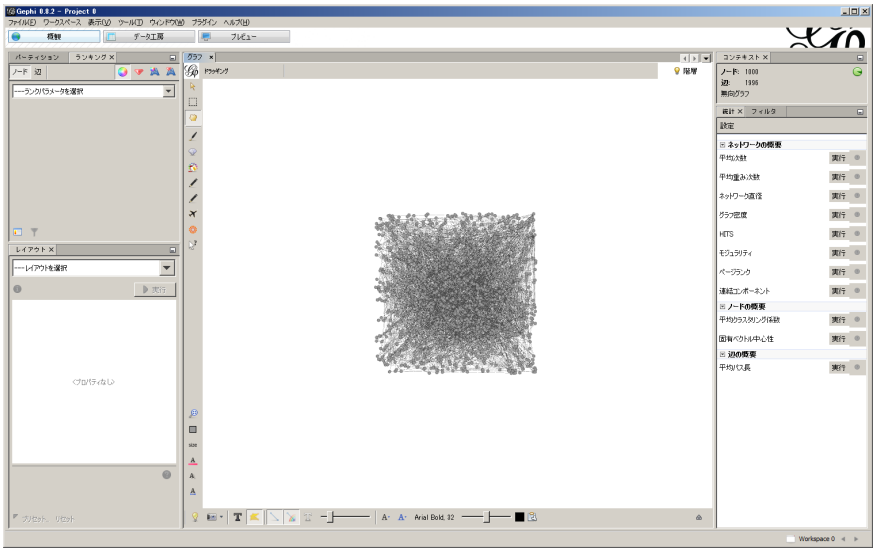


図 86: Gephi によるグラフの表示 (自動配置前)

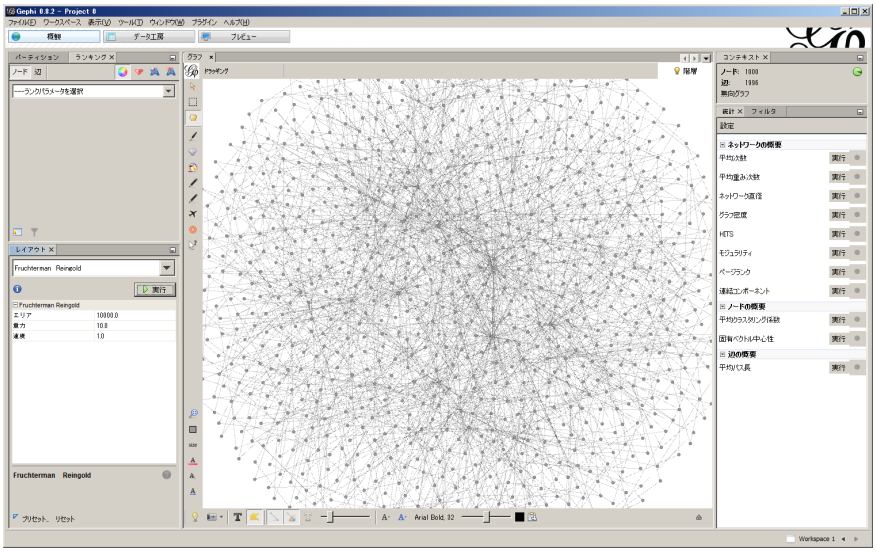


図 87: Gephi によるグラフの表示 (自動配置後)

8.15 GraphML フォーマットグラフの配置

S4 Simulation System に戻り、ブラウザパネルで「環境タブ」を選択してください。環境タブの「GraphML フォーマットグラフ」をドラッグし、モデル編集画面にドロップしてください。GraphML フォーマットグラフが配置されます。

モデル上の「GraphML フォーマットグラフ」をダブルクリックすると編集画面が表示されます。「グラフファイルの取り込み」ボタンを押すと、ファイルダイアログが表示されますので、先程 Gephi で自動配置後に保存した GraphML ファイルを選択して下さい。

8.16 モデルの再実行

モデル上の「非同期エージェント」をダブルクリックすると編集画面が表示されます。環境オブジェクトを、「eGraphMLFormatGraph(GraphML フォーマットグラフ)」に変更して下さい。

モデルメニューから「モデルを開始する」を選ぶと、モデルが実行されます。

以下のようなエージェントフレームが表示されます。(図 88)

9 ソーシャルフォースモデル

9.1 概要

ソーシャルフォースモデルは、群集行動の力学ベースモデルのひとつです。各歩行者は質量を持つ質点として表され、平面内で運動する粒子とみなします。各歩行者は、目的地を持ちますが、他の歩行者や、障害物から相互に干渉を受けながら、それぞれが運動するようなモデルでです。

S4 Simulation System では、ソーシャルフォースモデル用のエージェントが用意されていますので、障害物のある空間と、その経路点を定義しておけば、その空間上に歩行者を配置する事により、群集行動をシミュレートする事ができます。

9.2 プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「交差点」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモ

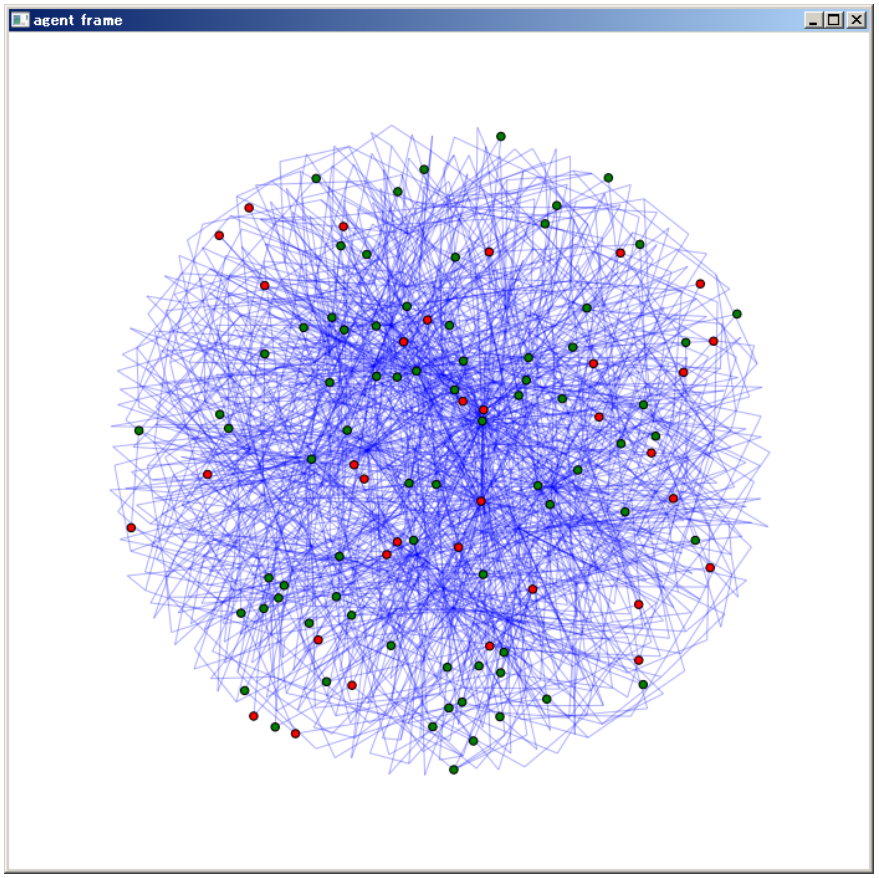


図 88: ツイートの伝播結果

デルが表示されます。次に、交差点シミュレーションモデルを作成していきます。

9.3　SFM エージェントの作成

ブラウザパネルで「エージェントタブ」を選択してください。エージェントタブの「SFM エージェント」をドラッグし、モデル編集画面にドロップしてください。SFM エージェントが配置されます。

9.4　SFM 環境の作成

ブラウザパネルで「環境タブ」を選択してください。環境タブの「SFM 環境」をドラッグし、モデル編集画面にドロップしてください。SFM 環境が配置されます。

9.5　SFM 環境の設定

「SFM 環境」をダブルクリックすると編集画面が表示されます。

様々な SFM のパラメータを設定します。このパラメータによって、エージェント間や他の障害物間の環境の具合がシミュレートされます。

ここでは、それぞれの値を以下のように設定して下さい。

パラメータ名	値
x0	0.0
x1	20.0
y0	0.0
y1	20.0
相互作用の強さ (N)	200.0
相互作用の範囲 (m)	0.2
弾性係数 (kg/s2)	120000.0
散逸係数 (kg/ms)	24000.0
最大影響半径 (m)	3.0
経路再探索間隔 (s)	15.0
最適速度 (m/s)	0.6
最高速度 (m/s)	1.5
歩行者の半径 (m)	0.2
加速時間 (s)	0.5
体重 (kg)	50.0
外力の変動係数	0.1
歩行者半径表示	表示
速度ベクトル表示	表示
歩行者間外力表示	非表示
障害物外力表示	非表示
経路グラフ表示	表示

9.6　環境の初期化後の処理

「SFM 環境」の属性設定タブにて、「環境の初期化後の処理」の「編集」をクリックします。すると、タブが開かれますので、そのタブ内で、以下のようなコードを入力します。

古いコードを消して、以下のコードに差し替えて下さい。

環境の初期化後の処理

```
self.setSFMMode(2)

# 障害物作成
rx = (self.x1 - self.x0) / 50.0
ry = (self.y1 - self.y0) / 50.0
def addobs(x, y):
    self.addObstacle(([x - rx / 2.0, y - ry / 2.0],
                      [x + rx / 2.0, y - ry / 2.0],
                      [x + rx / 2.0, y + ry / 2.0],
                      [x - rx / 2.0, y + ry / 2.0]))
addobs(self.x0 + rx * 5, self.y0 + ry * 5)
addobs(self.x1 - rx * 5, self.y1 - ry * 5)

# 経路ポイント作成
r = 1
self.left = self.addPathPoint(self.x0 + r, (self.y0 + self.y1) / 2.0, r)
self.right = self.addPathPoint(self.x1 - r, (self.y0 + self.y1) / 2.0, r)
self.top = self.addPathPoint((self.x0 + self.x1) / 2.0, self.y1 - r, r)
self.bottom = self.addPathPoint((self.x0 + self.x1) / 2.0, self.y0 + r, r)

# 経路ポイントを接続
self.addPathEdge(self.left, self.right)
self.addPathEdge(self.top, self.bottom)
```

このコードでは、障害物を定義した後、4 つの経路地点 (left, right, top, bottom) を定義しています。

経路地点の登録には、addPathPoint メソッドを利用します。addPathPoint(x, y, r) は座標 (x, y) に半径 r の経路地点を作成しまし、その経路地点番号を返します。(注意、経路エリアは障害物や他の経路エリアと重なってはいけません。重なった経路エリアは登録されずに、None を返します。)

次に、左右、上下の地点を接続し、経路としています。

経路地点の接続には、addPathPoint メソッドを利用します。addPathEdge(u, v) は、経路地点 u と 経路地点 v を接続します。

9.7　エージェントの環境の設定

「SFM エージェント」をダブルクリックすると編集画面が表示されます。

まず、エージェントと環境を結び付けます。「環境オブジェクト」のプルダウンメニューで、「eSFMEnvironment(SFM 環境)」を選択して下さい。

9.8　エージェント集合の初期化処理

「SFM エージェント」の属性設定タブから、「エージェント集合の初期化処理」を開いて下さい。

最初に、 `if len(self.peopleflows) == 0:` とある行の手前に、以下を挿入してください。

エージェント集合の初期化処理

```
# 起点と終点の組み合わせと重み
odpairs = [
    # (重み, (起点, 終点, 色))
    (1.0, (self.env.left, self.env.right, "r")),
    (1.0, (self.env.right, self.env.left, "c")),
    (1.0, (self.env.top, self.env.bottom, "b")),
    (1.0, (self.env.bottom, self.env.top, "y"))]

godpair = empiricalDistribution(odpairs)
```

次に、 `if len(self.peopleflows) == 0:` とある行の下にある

エージェント集合の初期化処理 (続き)

```
            vs = self.env.getAllPathPoints()
            # スタート経路地点
            v = next(sample(vs))
            generateAgents_(
                self.env.sampleInnerPathPoint(v),
                v,
                next(sample(vs)))
```

を、以下の通り置き換えてください。

エージェント集合の初期化処理 (続き)

```
            (src, dst, color) = next(godpair)
            generateAgents_(
                self.env.sampleInnerPathPoint(src),
                None,
                dst)
```

このコードは、各経路地点に、反対側に移動するようなエージェントをランダム発生さます。

ランダムな各経路地点と反対側の経路地点のペアを求めるには、先に、起点と終点の組み合わせを定義した `odpairs` を定義しています。この例で

は重みは一定なので、通常のサンプリングでも良いのですが、ここでは、empricalDistribution を使って、重み付きサンプリングを使っています。また、同時に起点の終点の組み合わせだけでなく、表示色も同時に定義しています。

generageAgents_ を呼ぶと、SFM に従うエージェントを作成します。

作成後は、SFM モデルが自動的に処理を行いますので、この例の場合は、特に何も処理しなくても自動的に終点に移動します。他のエージェントと相互作用した場合も、SFM に設定したパラメータによって、自動的に処理されます。

9.9　環境上のエージェント描画処理

「SFM 環境」をダブルクリックすると編集画面が表示されますので、の属性設定タブから、「環境上のエージェントの描画処理」を開いて下さい。

速度ベクトル表示の部分だけ、以下のように書き換えて下さい。

環境上のエージェントの描画処理

```
# 速度ベクトル表示
if self.dispV:
    for agent in agents:
        v = agent.v * 1
        screen.arrow(agent.p[0], agent.p[1],
                     v[0], v[1],
                     color = agent.screenColor,
                     edgecolor = agent.screenColor)
```

ここでは、あまり、本質的ではありませんが、エージェントを区別しやすいように、エージェントの速度ベクトルの色をエージェントの色に変えています。

9.10　パラメータの編集

次にモデルパラメータを設定します。モデルメニューから「パラメータを編集する」を開き、シミュレーション終了時間を 300 に設定します。

9.11　モデルの実行

モデルメニューから「モデルを開始する」を選ぶと、モデルが実行されます。

左から右には、赤色のエージェントが移動します。右から左には、水色のエージェントが移動します。上から下には、青色のエージェントが移動します。下から上には、黄色のエージェントが移動します。

本モデルでは、

1. 経路地点を 4 つ (左、右、上、下) 定義した。
2. 経路を 2 つ (左右、上下) 定義した。
3. 各経路地点に、反対側に移動するようなエージェントをランダム発生させた。

とするだけで、交差部分で発生するエージェントの相互作用をシミュレートする事ができました。

10 施設内人流モデル

10.1 概要

ここでは、ソーシャルフォースモデルの具体例として、施設内を通行する人流シミュレーションモデルを取り上げます。作成するモデルは下記のようなモデルです。

- 施設内には入口と出口がある。
- エージェントは入口から施設に入り、出口から出ていく。

S4 Simulation System では、ソーシャルフォースモデル用のレイアウトを作成するための、SFM 地図エディタが備わっています。SFM 地図エディタを使えば、施設のレイアウトから出入り口の設定までを GUI で簡単に作成することができます。

10.2 プロジェクトの作成

S4 Simulation System を起動し、ファイルメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「施設内人流シミュレーション」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。

10.3　環境部品の作成

エージェントが振舞うシミュレーション空間を作成します。ソーシャルフォースモデルにおけるシミュレーション空間の構成要素とモデルとの対応は以下になります。

シミュレーション空間	施設
障害物 (エージェントが通過できない場所)	壁
エージェントが生成される地点	出入り口
経路地点 (エージェントが通過する可能性のある地点)	通路
エージェントが消滅する地点	出入り口

10.4　SFM 地図の配置

ブラウザパネルで「環境タブ」を選択してください。環境タブの「SFM 地図」をモデル編集画面 (右側) にドラッグ&ドロップしてください。

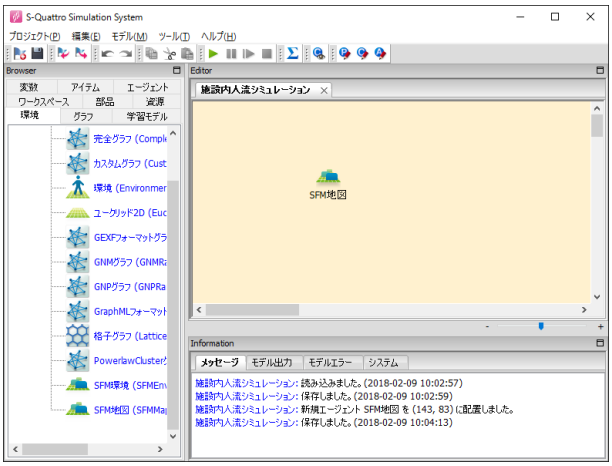


図 89: SFM 地図の配置

10.5　SFM 地図エディタの起動

SFM 地図エディタを使って、SFM 地図部品にレイアウトを作成していきます。「SFM 地図」をダブルクリックし、設定画面を表示させます。地図編集ボタンをクリックすると、SFM 地図エディタが起動します。(図 90)

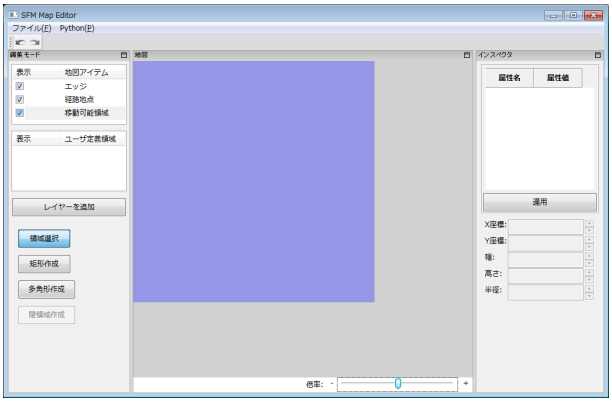


図 90: SFM 地図エディタ

10.6　レイアウトの作成

ここではレイアウトの要素である、移動可能な空間や、移動不可能な壁を作成します。SFM 地図エディタでは、多角形を組み合わせる事で表現をします。まず初めに、下記の手順で移動可能な空間を作ります。

- ファイルメニューから、地図初期化をクリックします。
- 横: 100、縦: 100 を設定します。(図 91)
- 倍率を調整するか、エディタのサイズを調整し、描画領域 (青い領域) がすべて収まるようにします。
- 編集モード（画面左上）の移動可能領域と書かれた箇所をクリックし、領域操作モードにします
- 矩形作成ボタンをクリックします。
- 空間 (移動可能) をクリックします。
- 地図エディタ上で左ボタンを押しながら、マウスを操作し、左ボタンを離します。
- 上記操作を繰り返し、多角形を組み合わせていきます。(図 92)

空間サイズの設定ウィンドウでは、横と縦のほかに空間参照系と画面中心緯度・経度を設定できます。空間参照系は地図エディタの座標を緯度経度に変換する規則を表します。画面中心緯度・経度は現在の空間参照系の元で画面中心が現実のどの場所に対応するかを緯度経度で表します。

次に、組み合わせた多角形を一つの領域にし、空間を作成します。

- 領域選択ボタンをクリックします。

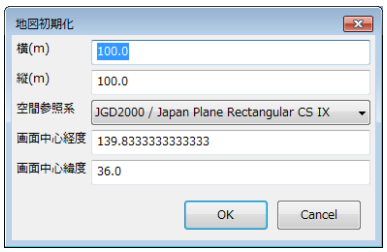


図 91: 空間サイズの設定

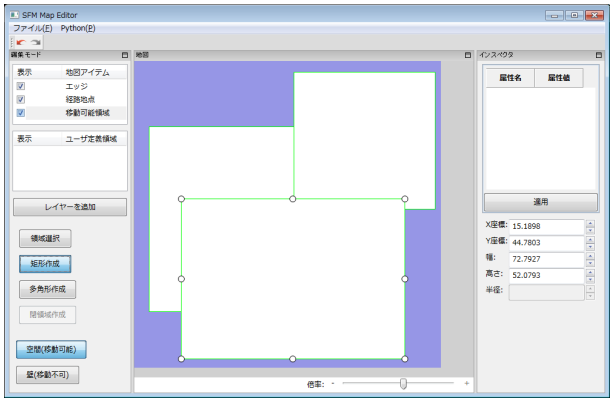


図 92: 移動可能な空間の作成

- 地図エディタ上で左ボタンを押しながらマウスを操作し、一つの領域にしたい多角形を選択し、左ボタンを離します。
- 選択した領域上で、右クリックし、グループ化を選択します。(図 93)

同様に壁を作成するには、下記の手順を踏みます。

- 編集モード（画面左上）の移動可能領域と書かれた箇所をクリックし、領域操作モードにします
- 矩形作成ボタンをクリックします。
- 壁 (移動不可) をクリックします。
- 地図エディタ上で左ボタンを押しながら、マウスを操作し、左ボタンを離します。
- 上記操作を繰り返し、壁を作成していきます (図 94)

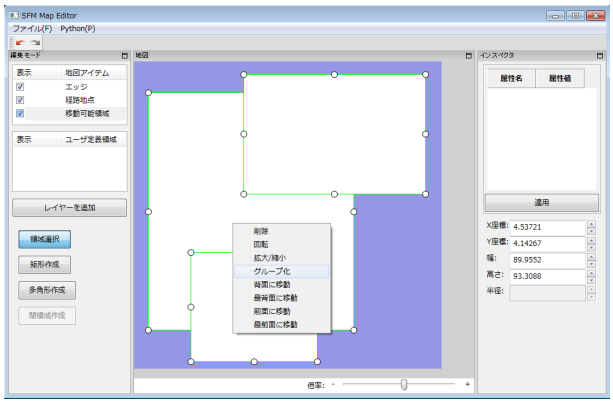


図 93: グループ化

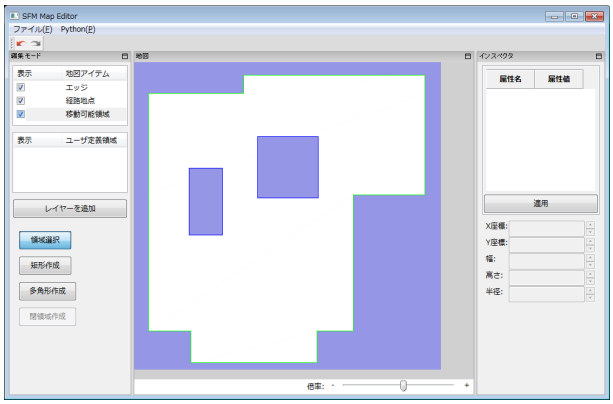


図 94: 壁の作成

10.7　経路地点の作成

ここではエージェントが通過する可能性のある地点である、経路地点を作成します。経路地点を作成するには、下記の手順を踏みます。

- 編集モード（画面左上）の経路地点と書かれた箇所をクリックし、経路地点モードにします
- 経路地点作成ボタンをクリックします。
- 地図エディタ上で経路地点を作成したい場所で、左クリックします
- 上記操作を繰り返し、経路地点を作成していきます

もしも誤って作成した場合には、経路地点選択ボタンを押して、削除したい経路地点をクリックします。その後、右クリックし、削除を選択します。(図95)

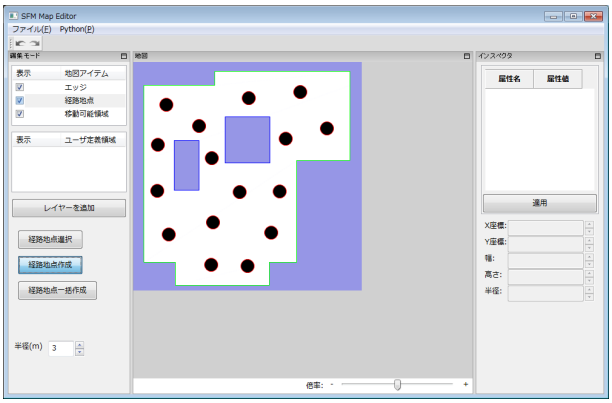


図 95: 経路地点の作成

10.8　エッジの作成

経路地点同士をエッジで連結するとエージェントは、連結した経路地点間を移動できるようになります。ここでは、以下の手順でエッジを連結します。

- 編集モード（画面左上）のエッジと書かれた箇所をクリックし、エッジ操作モードにします
- エッジ作成ボタンをクリックします。
- 地図エディタ上で連結したい経路地点上で左ボタンを押しながら、連結先の経路地点までマウスを移動させ、ボタンを離します。

- 上記操作を繰り返し、エッジを作成していきます

もしも誤って作成した場合には、エッジ選択ボタンを押して、削除したいエッジをクリックします。その後、右クリックし、削除を選択します。(図 96)

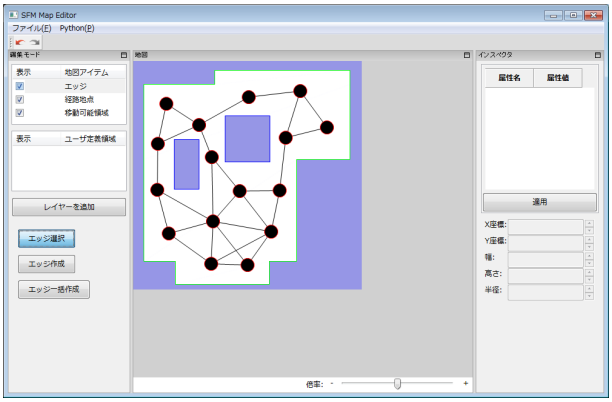


図 96: エッジの作成

連結するエッジが多い場合には、下記のようにして一括して作成することもできます。

- 編集モード（画面左上）のエッジと書かれた箇所をクリックし、エッジ操作モードにします
- エッジ一括作成ボタンを押します。
- 地図エディタ上で一括作成したい領域を選択します。
- 最大距離には、連結するエッジの最大距離を指定します

ただし、この方法ですと、壁を越えてエッジが結ばれてしまいます。そのようなエッジを一括して削除したい場合には、下記の手順を踏みます。

- エッジ選択ボタンを押します。
- 地図エディタ上で一括して削除したい領域を選択します。
- 右クリックでエラーエッジを選択します
- 右クリックで削除を選択します。

10.9　属性の設定

経路地点にエージェントが生成される地点 (start)、エージェントの目的地となる地点 (goal) を設定します。ここで設定した属性は、SFM エージェント

部品から参照することができますので、作成するモデルに応じて、必要な属性を設定しておきます。

- 編集モードを経路地点操作モードにした上で右クリックメニューから属性管理テーブルを表示を選択します。(図 97)
- 追加ボタンをクリックし、属性名に role を、デフォルト値に nothing を入力します。この操作によって、全てのノードに属性 role、値 nothing が設定されました。各ノードの属性値は、地図エディタ右上のインスペクタから確認することができます。(図 98)
- 属性管理テーブルを閉じます。
- 経路地点選択をクリックします。
- エージェントが生成される経路地点をクリックし、右側のインスペクタで属性値に start と入力します。

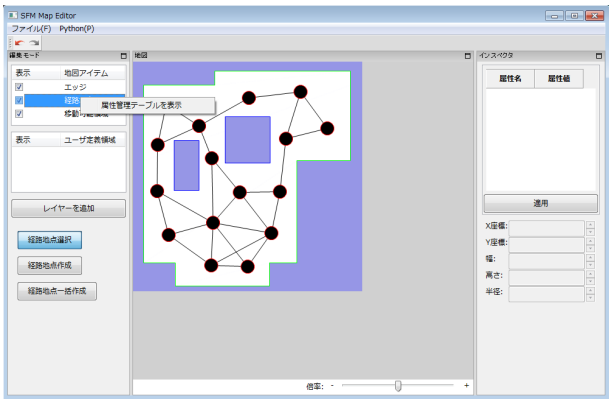


図 97: 属性管理テーブルを表示

上記を繰り返して、エージェントが生成される地点を設定します。エージェントの目的地の設定も同様です。その場合は、属性に goal と入力します。属性を設定出来たら、ファイルメニューから地図保存を選択し、地図エディタを閉じます。

10.10　SFM エージェントの配置

ソーシャルフォースモデルにおけるエージェントの定義をしていきます。ブラウザパネルで「エージェントタブ」を選択してください。エージェントタブの「SFM エージェント」をモデル編集画面 (右側) にドラッグ&ドロップしてください。

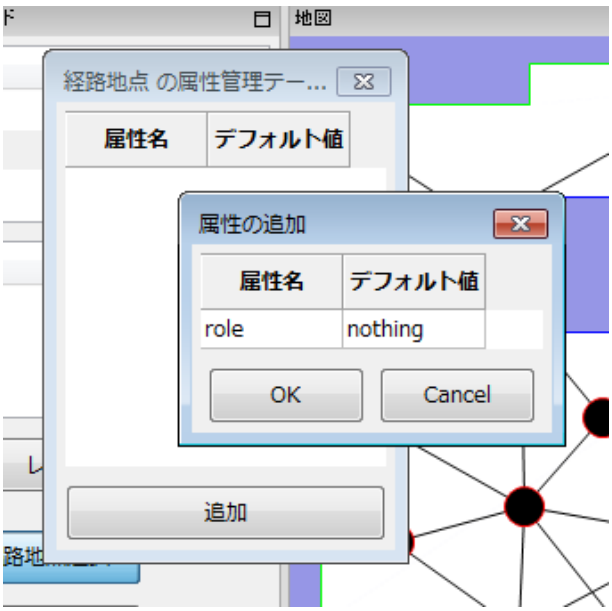


図 98: 属性の追加

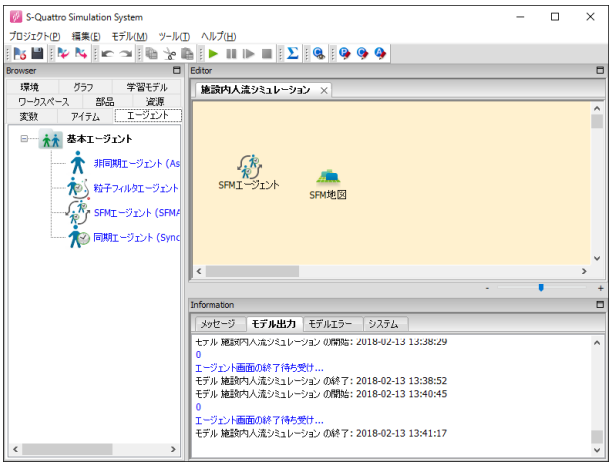


図 99: SFM エージェントの配置

10.11　SFM エージェントの設定

SFM 地図で設定したレイアウト上を SFM エージェントが振舞うようにするには、下記の手順を踏みます。

- エージェント部品をダブルクリックします。
- 環境オブジェクトに、SFM 地図を設定します。(図 100)

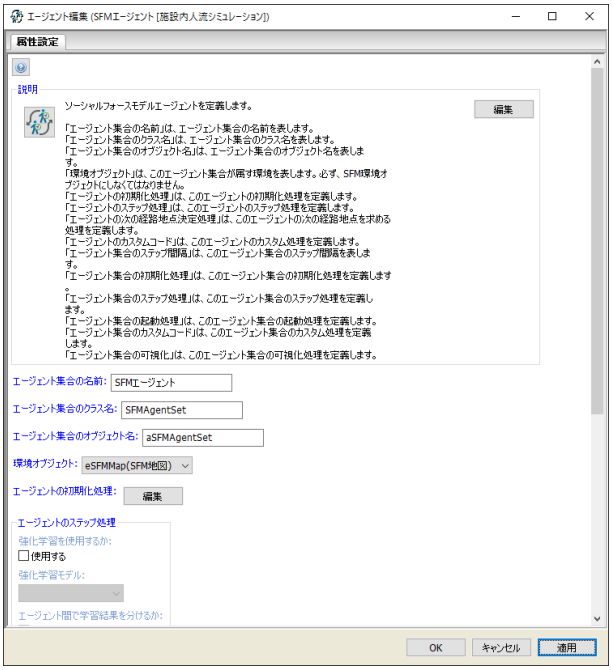


図 100: SFM エージェント部品

次に、SFM 地図で設定した属性 start の経路地点でエージェントが生成され、経路地点 goal に向かうように設定をします。

- エージェント部品をダブルクリックします。
- エージェント集合の初期化処理をクリックし、`if len(self.peopleflows) == 0:` の行の直前に以下の「start,goal の取得」で囲まれているプログラムを挿入します。
- `if len(self.peopleflows) == 0:` の行の下にある `v = next(sample(vs))` を `v = next(gStartnodes)` に、`next(sample(vs))` を `next(gGoalnodes))` に置き換え、`generateAgents_` に start, goal の経路地点を渡します。
  以下、「start,goal の取得」のプログラムの解説となります。

- SFM 地図のメソッドを使い、 start, goal 役のノードを取得します。また、 start, goal 役のノードからランダムに 1 つノードを取得するジェネレータを定義します。
- ノード内のランダムな座標を `sampleInnerPathPoint` メソッドで取得します。取得した座標は、`generateAgents_` の引数 p に渡します。
- 属性 goal のノードからランダムに 1 つノードを取得し、`generateAgents_` の引数 `goal` に渡します。(図 101)

start, goal の取得

```
pps = self.env.getAllPathPoints()
startnodes = [
    p for p in pps if p.getAttrs()["role"] == "start"]
gStartnodes = sample(startnodes)
goalnodes = [
    p for p in pps if p.getAttrs()["role"] == "goal"]
gGoalnodes = sample(goalnodes)
```

10.12　シミュレーションの実行

以上で設定は終了です。GUI のモデルメニューから「モデルを開始する」をクリックすると、シミュレーションが実行できます。(図 102)

10.13　3D アニメーションの表示

SFM 地図を使ったソーシャルフォースシミュレーションは、3D アニメーション部品を利用することで、3D アニメーション表示を行うことができます。

10.13.1　3D アニメーションの配置

ブラウザパネルで「アニメーションタブ」を選択してください。アニメーションタブの「3D アニメーション」をモデル編集画面 (右側) にドラッグ&ドロップしてください。

10.13.2　3D アニメーションの設定

3D アニメーション表示の対象となる SFM 地図への参照を、次の手順によって設定します。

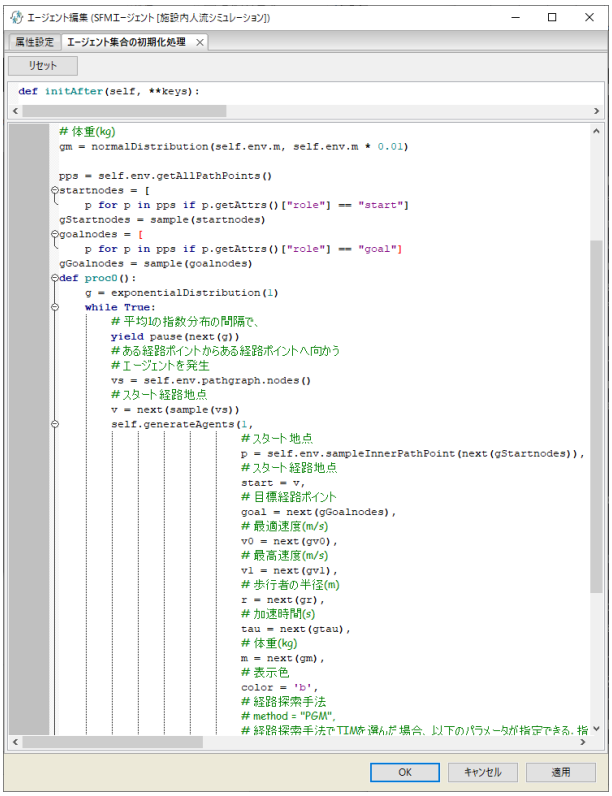


図 101: 経路地点の属性の参照

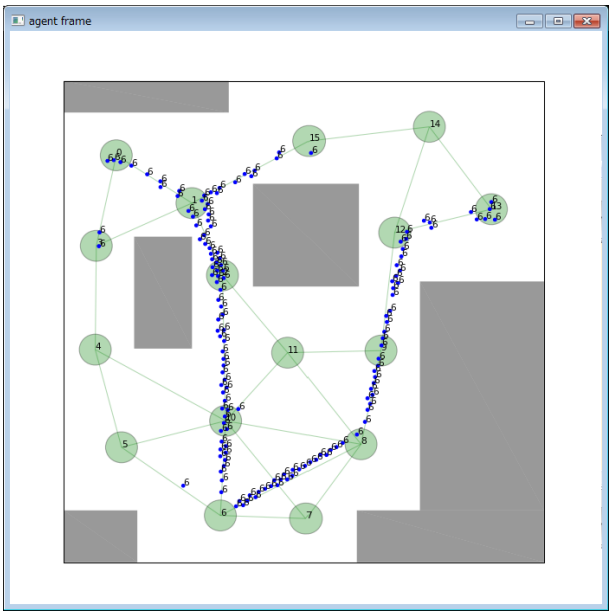


図 102: 経路地点の属性の参照

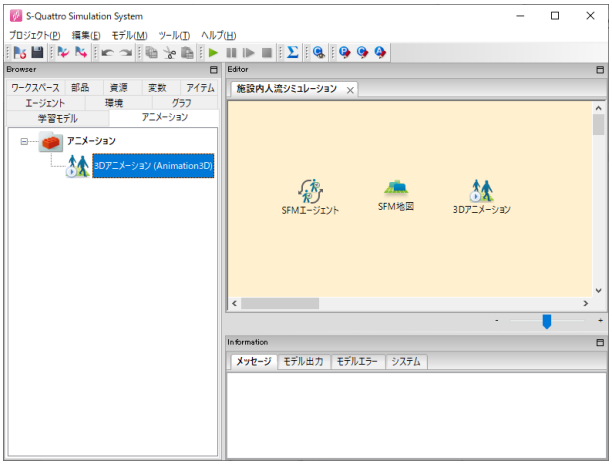


図 103: 3D アニメーションの配置

- 3D アニメーション部品をダブルクリックします。
- 環境オブジェクトに、SFM 地図を設定します。(図 104)

10.13.3　シミュレーションの実行

3D アニメーション表示用の出力データを作るためには、シミュレーションの実行が必要です。GUI のモデルメニューから「モデルを開始する」をクリックして、シミュレーションを実行します。シミュレーションが完了したら、GUI のモデルメニューから「モデルを停止する」をクリックして、シミュレーション実行画面を閉じます。

10.13.4　3D アニメーションの実行

3D アニメーション表示は、3D アニメーションビューワーによっておこなわれます。

- 3D アニメーション部品を右クリックし、コンテキストメニューを開きます。
- 3D アニメーション再生をクリックします。(図 105)
- 3D アニメーションビューワーが起動されます。(図 106)

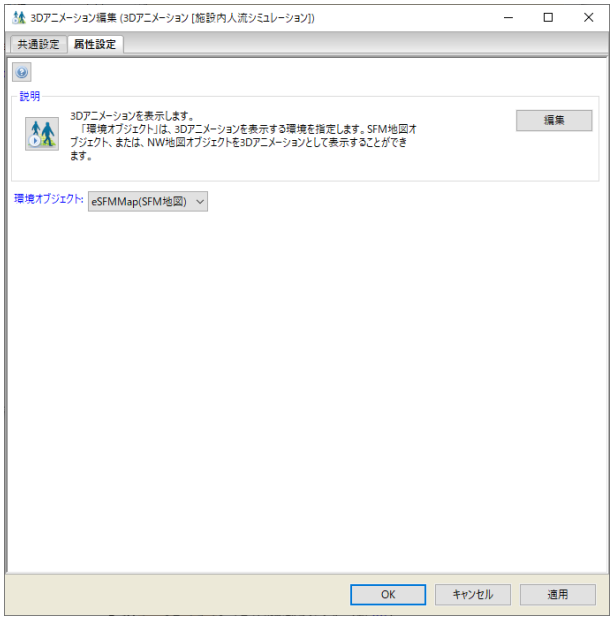


図 104: 3D アニメーション部品の設定

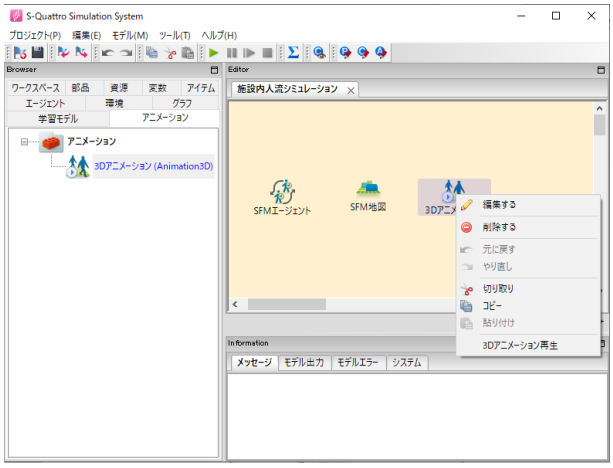


図 105: 3D アニメーション再生

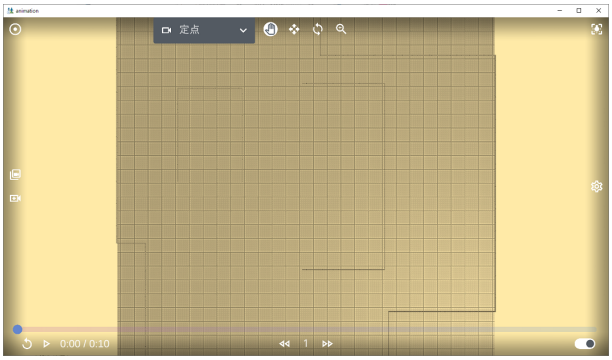


図 106: 3D アニメーションビューワー起動

10.13.5　3D アニメーションの視点切り替え

3D アニメーションビューワーが起動したとき、はじめは真上からの視点で表示されています。次の手順で視点を切り替えることができます。

- 画面左端のボタンから視点コレクションを展開します。
- 「カメラ 2」を選択します。(図 107)

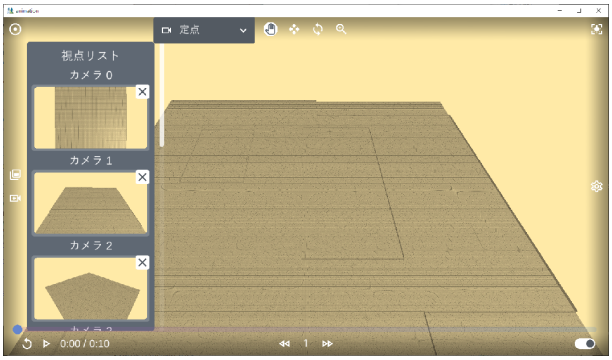


図 107: 3D アニメーションの視点切り替え

このようにすることで前方斜め 45 度からの視点に切り替えることができます。ドロップダウンメニューには視点の位置が記録されており、初期値として次のものが用意されています。

- カメラ 1：真上から
- カメラ 2：前方斜め 45 度から
- カメラ 3：右前方斜め 45 度から
- カメラ 4：右斜め 45 度から

- カメラ 5 : 右後方斜め 45 度から
- カメラ 6 : 後方斜め 45 度から
- カメラ 7 : 左後方斜め 45 度から
- カメラ 8 : 左斜め 45 度から
- カメラ 9 : 左前方斜め 45 度から

10.13.6　3D アニメーションの視点操作

3D アニメーションビューワーではマウスによって視点を操作することができます。

視点の空間的な位置のことをカメラと呼び、画面の中央をターゲットと呼びます。よって、視点はカメラとターゲットの位置にしたがって決められます。

操作ツールとして、次のものが用意されています。それぞれのツールを選択して、マウスクリックおよびマウスドラッグすることで視点操作が可能です。

- ハンドツール：エージェントの選択および選択解除
- 移動ツール：ターゲットの移動
- 回転ツール：ターゲットを中心に視点の回転
- 拡大ツール：ターゲットへズームイン、ズームアウト

また、マウススクロールでもズームイン、ズームアウトを行うことができます。

カメラモードとして、つぎの三つが用意されています。

- 定点カメラ : 自由に視点移動ができる固定視点のカメラです。
- ターゲットカメラ : 特定のエージェントをターゲットとして距離を保ちながら移動するカメラです。
- 一人称視点カメラ : ターゲットエージェントの視野を映すカメラです。

エージェントの切り替えは、つぎのキー操作が対応しています。

- ↑ または ↓ : ターゲットエージェントの切り替え

10.13.7　3D アニメーションの再生操作

3D アニメーションの再生に関する操作は画面下部のボタンやスライダーによっておこなうことができます。

画面下部のボタン、スライダーと機能は画面左から順に次のような対応になっています。

- はじめに戻る
- 再生、一時停止
- 再生時間スライダー
- 再生速度を遅くする
- 再生速度を速くする
- UI 表示/非表示切り替え

10.13.8　3D アニメーションの環境設定

3D アニメーションの環境設定画面で、背景色や床・壁の素材、および壁の高さを変更することができます。設定を変更するには、次の手順をおこないます。

- 画面右端の環境設定ボタンをクリックします。
- 環境設定画面上で、変更したい項目のタブを選び、素材アイコンをクリックします。図 108
- クリックした素材が反映されます。

また、壁の高さスライダーを動かすことで壁の高さを変えることができます。

10.14　可視化ダッシュボードの作成

SFM 地図、SFM 環境、NW 地図を使ったエージェントシミュレーションは、シミュレーション後に可視化用のダッシュボードを作成できます。

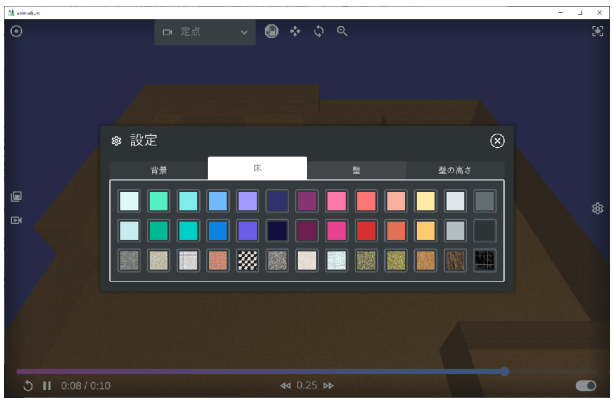


図 108: 3D アニメーションの環境設定

10.14.1　可視化用データの出力設定

可視化ダッシュボード機能を利用するには、起動メニューからデータの出力設定を行った後、シミュレーションを実行することが必要です。

起動メニューは SFM 地図の編集画面にあります（図 109）。可視化の記録を「あり」、間隔を例えば 1 と設定して OK ボタンを押します。

その後、シミュレーションを実行することで、エージェントの 1 秒置きの軌跡データがプロジェクトフォルダ（data/XXX_record/以下）に出力されます。

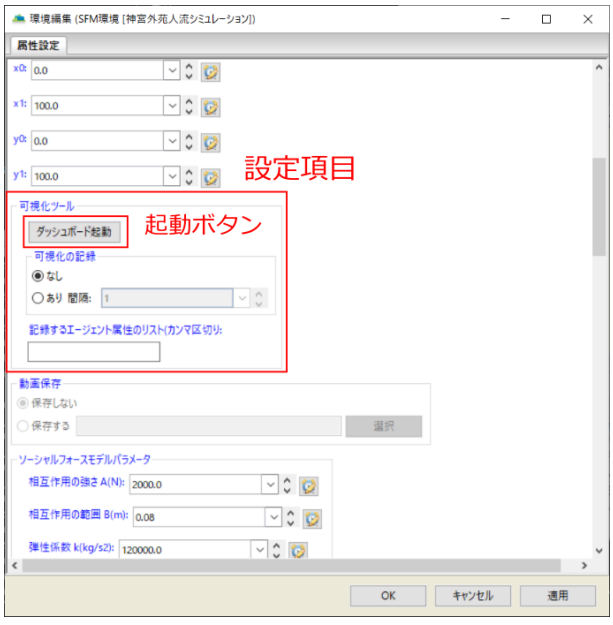


図 109: 可視化ダッシュボード起動メニュー

10.14.2　ダッシュボードの起動

起動メニューの中のダッシュボード起動ボタンをクリックすると、コマンドプロンプトが立ち上がった後、お使いのパソコンで既定のブラウザとして設定されているブラウザが自動的に立ち上がります。

コマンドプロンプトを閉じるとサーバーが停止しますので、ダッシュボード使用中は閉じずに置いておきます。

10.14.3　プロットコンポーネント

追加の横のドロップダウンがプロットになっていることを確認して、その横の＋ボタンをクリックすると、グリッドパネルにプロットコンポーネントが追加されます。

プロットコンポーネントのスライダーの値を変更することで、エージェントの軌跡データを可視化することができます。

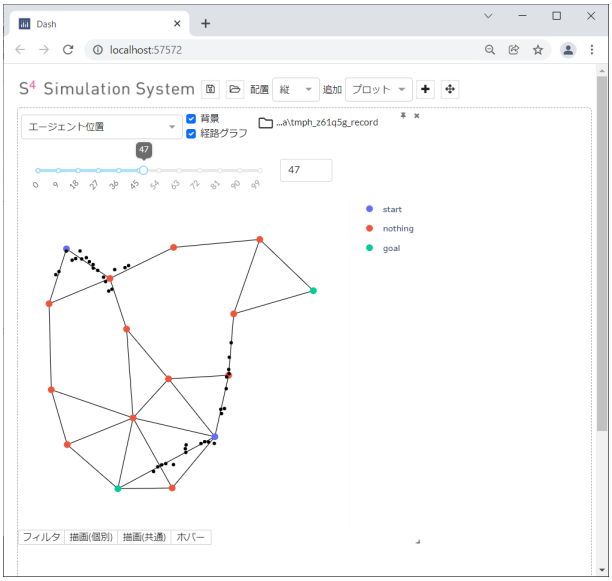


図 110: 可視化ダッシュボード

左下のタブをクリックすると、プロットの詳細設定を行うことができます。例として、描画（個別）をクリックして、経路地点色に「role」を選択します。次に描画（共通）をクリックして、経路地点サイズを「10px」に変更し、もう一度描画（共通）をクリックしてタブを閉じます。そうすると先に設定した start、goal および nothing が可視化され、エージェントの動きがより理解しやすくなります（図 110）。

プロットコンポーネントの左上にあるドロップダウンの値を変更することで、プロットの種類を変更することができます。また、プロットコンポーネ

ントの右上にあるフォルダアイコンをクリックすると、ファイル選択ダイアログが開きます。他プロジェクトの data/XXX_record 内にある path_info.txt を選択することで、プロット内容をそのプロジェクトのものに切り替えることができます。

10.14.4　テキストコンポーネント

同様にテキストコンポーネントを配置することもできます。追加の横のドロップダウンがテキストになっていることを確認して、その横の+ボタンをクリックします。２つ目以降のコンポーネントについては、+ボタンの横のドラッグ要素をドラッグ&ドロップすることによっても追加可能です。

テキストコンポーネントにマークダウンおよび数式 ($または$$で囲う) を入力し、左下の三角ボタンからメニューを開いて表示ボタンをクリックすると、それらがレンダリングされて表示されます。それぞれ CommonMark、KaTeX に準ずるものを記述してください。

10.14.5　グリッド操作

各コンポーネントの左上もしくは中央上にカーソルを乗せると、ドラッグ要素が描画されます。ドラッグ要素をドラッグすることでコンポーネントを移動させることができます。

右上にはピンボタンおよび削除ボタンが描画されています。ピンボタンを押すと、位置および中身が固定されます。削除ボタンを押すと、確認ダイアログが表示されたのち、コンポーネントを削除することができます。

右下には拡大・縮小ハンドルが描画されています。ハンドルをドラッグすることでグリッドの大きさを変更することができます。

10.14.6　ダッシュボードの保存・読み込み

ダッシュボードが作成できたらメニュー左端のセーブボタンを押すことで、ダッシュボードを pickle ファイルとして保存することができます。

その隣のロードボタンから pickle ファイルを選択、または pickle ファイルをロードボタンにドラッグ&ドロップすることで、保存されたダッシュボードを読み込むことができます。

10.14.7　ダッシュボードの終了

最初に立ち上がったコマンドプロンプトを閉じることで、ダッシュボードのサーバーが停止します。そのあとブラウザを閉じます。

11　映画館人流モデル

11.1　概要

ここでは、ソーシャルフォースモデルのより高度な機能を用いたシミュレーションモデルとして、映画館における観客の退出行動のシミュレーションを行います。

11.2　プロジェクトの作成

S4 Simulation System を起動し、プロジェクトメニューから「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、プロジェクト名を「映画館人流シミュレーション」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。モデルをダブルクリックすると、モデル編集パネルに空のモデルが表示されます。

11.3　SFM 地図の配置

ブラウザパネルで「環境タブ」を選択してください。環境タブの「SFM 地図」をモデル編集画面 (右側) にドラッグ&ドロップしてください。

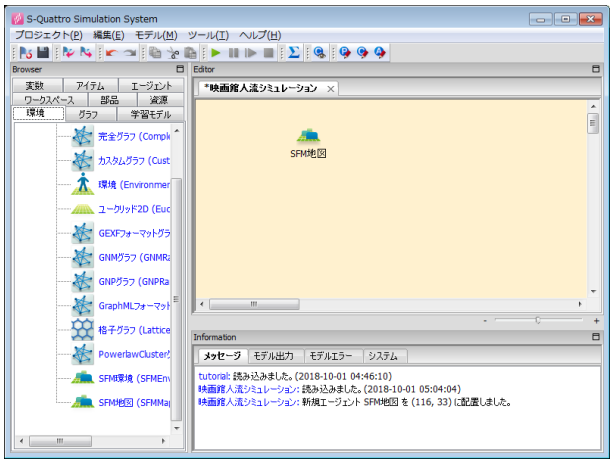


図 111: SFM 地図の配置

11.4　SFM 地図エディタの起動

SFM 地図エディタを使って、SFM 地図部品にレイアウトを作成していきます。「SFM 地図」をダブルクリックし、設定画面を表示させます。地図編集ボタンをクリックすると、SFM 地図エディタが起動します。(図 112)

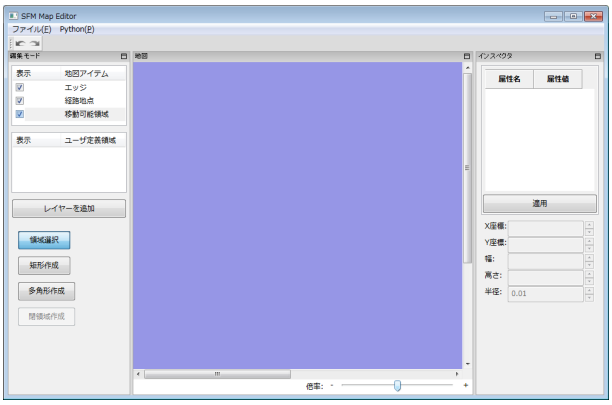


図 112: SFM 地図エディタ

11.5　地図の初期化

- ファイルメニューから、地図初期化をクリックします。
- 横: 20、縦: 25 を設定します。(図 113)
- 倍率を調整するか、エディタのサイズを調整し、描画領域 (青い領域) がすべて収まるようにします。

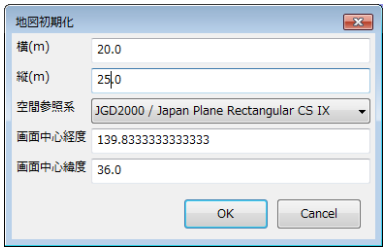


図 113: 空間サイズの設定

11.6　背景画像の取り込み

予め用意した映画館内部の見取り図を背景画像として取り込み、その上に経路地点などを配置していきます。まず、ファイルメニューから、背景 PNG

読み込みをクリックします。(図 114) (S-Quattro Simulation System のインストールフォルダ)￥samples￥theater.png を選択します。映画館内部の見取り図が表示されます。(図 115) （S-Quattro Simulation System のインストールフォルダ) は、デフォルトでは下記になります。

C:￥Program Files￥Mathematical Systems Inc￥S-Quattro Simulation System V6

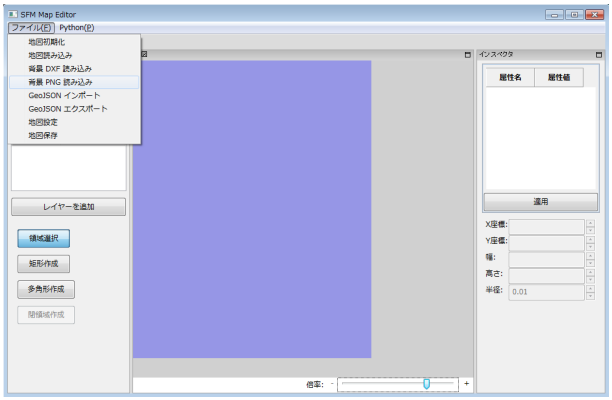


図 114: png を読み込み

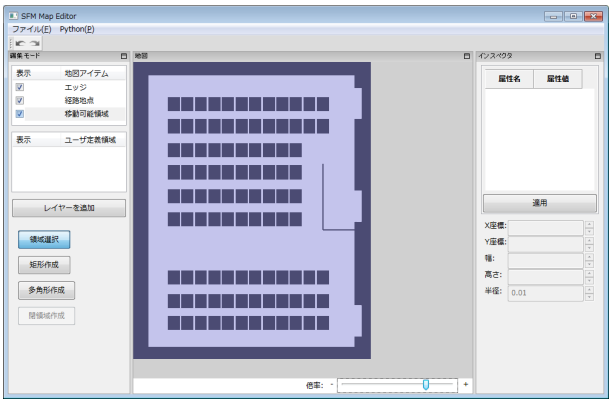


図 115: png を読み込み

11.7　領域の作成

ここではレイアウトの要素である、移動可能な空間や、移動不可能な壁を作成します。映画館の背景画像をなぞって空間と壁を配置していきます。

- 先に壁を作成してから空間を作成します。先に空間を作成してしまうと、壁としてなぞりたい背景画像の部分が隠されてしまいます。

- 壁は各席の背中側と中央出入り口のスロープを区切るように作成します。(図 116)
- 空間は部屋全体に対応するものが 1 つと、出入口に対応するものを 3 つ作成し、グループ化します。壁が隠れてしまうため、右クリックメニューから最背面に移動を選択し、作成したグループの上に壁が現れるようにします。(図 117)

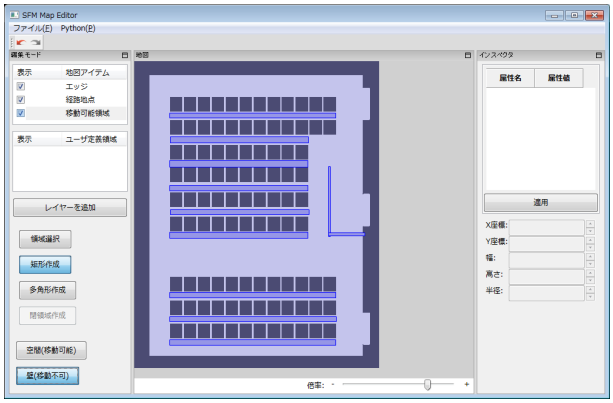


図 116: 壁の配置

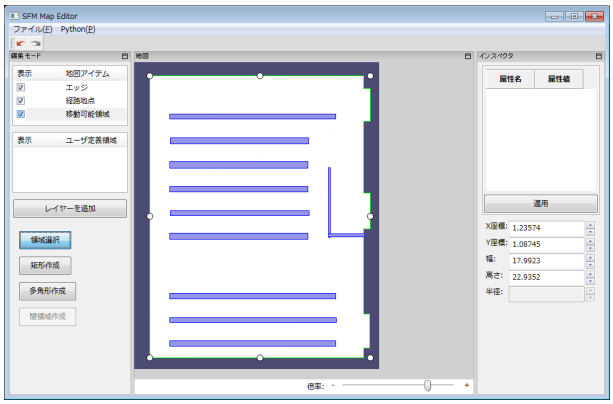


図 117: 空間の配置

11.8　属性の設定

ここでは次に作成する客席などの経路地点の属性をあらかじめ設定しておきます。

- 編集モードを経路地点操作モードにした上で右クリックメニューから属性管理テーブルを表示を選択します。
- 追加ボタンをクリックし、属性名に area を、デフォルト値に nothing を入力します。
- 追加ボタンをクリックし、属性名に role を、デフォルト値に nothing を入力します。
- 属性管理テーブルを閉じます。

11.9　経路地点の作成

ここでは観客が座っている客席を表現する経路地点、および出入口を表現する経路地点を作成します。

客席は手動で配置することも可能ですが、数量が多いため実際には時間がかかってしまいます。ここでは Python コンソール機能を利用して自動的に客席を配置します。まず Python メニューから、Python コンソールを開くを選択します。その後、客席（経路地点）を配置するスクリプトを記述していきます。客席は縦横の方向ごとに等間隔で長方形状に並んでいるため、for ループにより規則的に配置することができます。API による操作（ここでは addPathPoint）は手動の場合と同様に undo 記録が取られるため、一度に大量の経路地点を置く場合は動作が遅くなります。これに対し undo 記録など余計な処理をスキップして速度低下を回避する AtOnce という API が用意されています。実行するスクリプトは以下になります。

経路地点の自動配置

```
def putSeats(px, py, qx, qy, nr, nc, skips = None, radius = 0.3, attrs = None):
    for r in range(nr):
        s = float(r) / (nr - 1)
        y = (1 - s) * py + s * qy
        for c in range(nc):
            if skips is not None and c in skips[r]:
                continue
            t = float(c) / (nc - 1)
            x = (1 - t) * px + t * qx
            addPathPoint(x, y, radius, attrs)
removePathPoints(getAllPathPoints())
skips = [[],[],[10,11],[10,11],[10,11],[10,11]]
with AtOnce():
    a1 = {"area": "a1", "role": "seat"}
    putSeats(3.45,  3.55, 15.95, 13.25, 6, 12, skips=skips, attrs=a1)
with AtOnce():
    a2 = {"area": "a2", "role": "seat"}
    putSeats(3.45, 18.10, 15.95, 21.95, 3, 12, attrs=a2)
```

一度に全体をコピー&ペーストするとインデントが崩れ文法エラーとなります。コピー&ペーストする場合は、インデントに注意しながら 1 行ずつコピー&ペーストしてください。

客席は以下のように配置されます。(図 118)

出入口は手動で配置します。配置後、右側のインスペクタで role の属性値を exit にします。(図 119)

11.10　経路計算手法の設定

SFM 地図の設定より、経路計算手法として”TIM” もしくは”’CMM’ を選択します。

11.11　SFM エージェントの配置

ソーシャルフォースモデルにおけるエージェントの定義をしていきます。ブラウザパネルで「エージェントタブ」を選択してください。エージェントタブの「SFM エージェント」をモデル編集画面 (右側) にドラッグ&ドロップしてください。

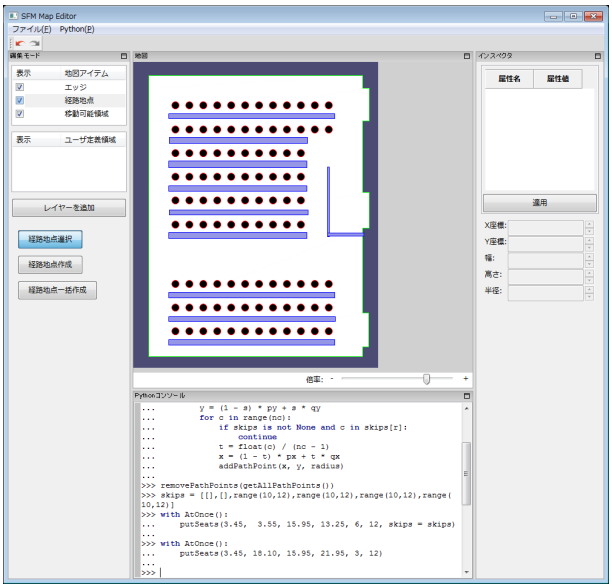


図 118: 客席の配置

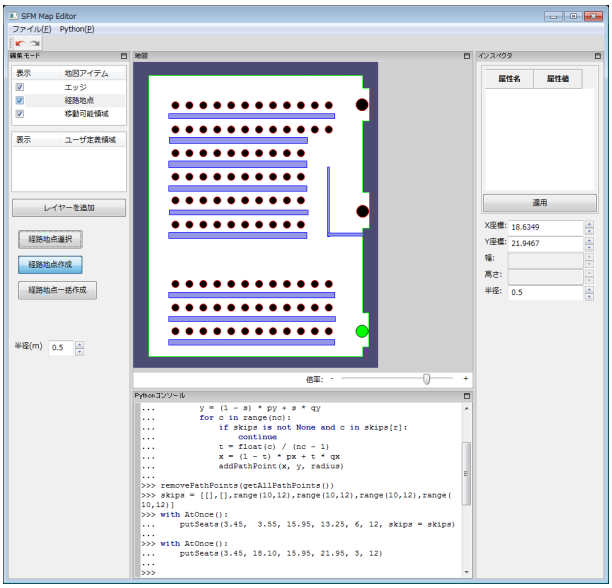


図 119: 出入口の配置

図 120: 経路計算手法の設定

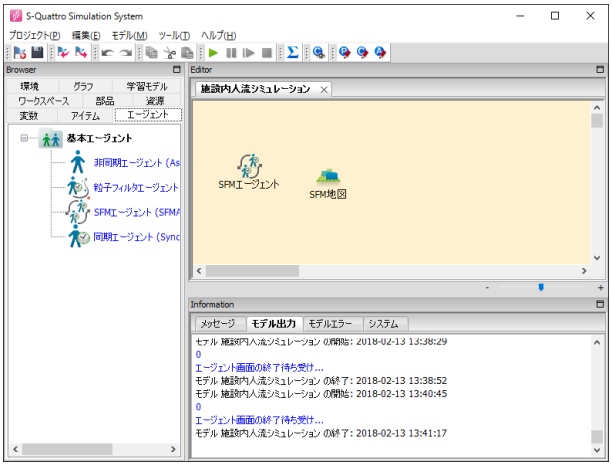


図 121: SFM エージェントの配置

11.12　SFM エージェントの設定

SFM 地図で設定したレイアウト上を SFM エージェントが振舞うようにします。最初に、SFM 地図への参照を設定します。

- エージェント部品をダブルクリックします。
- 環境オブジェクトに、SFM 地図を設定します。(図 122)

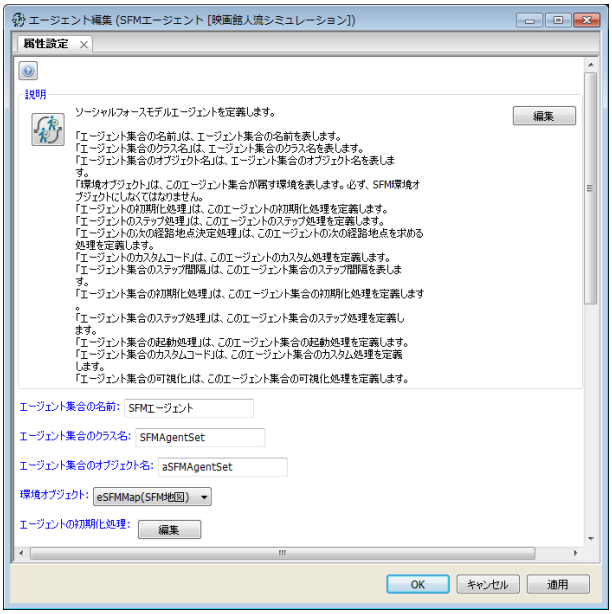


図 122: SFM エージェント部品

今回エージェントに施す設定の内容は以下の通りです。

- 観客が客席に座っている状態からシミュレーションが開始します。客席の利用率は 80% です。
- 2 つのエリア a1, a2 が個別に退出指示を待ち、指示を受けたら退出行動に入ります。まずシミュレーション開始時にエリア a1 に退出指示を出し、70%が退出した時点でエリア a2 に退出指示を出します。
- 退出指示を受けた瞬間にではなく、指数分布で平均 5 秒待ってから移動を開始します。
- すべての観客が中央の出入口から退出します。

実際に設定を施していきます。

エージェント部品の編集画面からエージェント集合の初期化処理を開き、`if len(self.peopleflows) == 0:` 以下の行をすべて削除して、以下のコードで置き換えます。

注意: 環境上のエージェント集合の初期化処理 (1) とエージェント集合の初期化処理 (2) とエージェント集合の初期化処理 (3) に分かれていますが、すべてを「エージェント集合の初期化処理」内につなげて入力します。インデントの位置にご注意ください。また、このモデルと同じモデルが下記にあります。

（S-Quattro Simulation System のインストールフォルダ)￥samples￥映画館人流シミュレーション.s4

プロジェクトメニューからインポートしてコピー&ペーストされる際にご利用ください。

（S-Quattro Simulation System のインストールフォルダ) は、デフォルトでは下記になります。

C:￥Program Files￥Mathematical Systems Inc￥S-Quattro Simulation System V6

エージェント集合の初期化処理 (1)

```
pps = self.env.getAllPathPoints()
startnodes = [
    p for p in pps if p.getAttrs()["role"] == "seat"]
goalnodes = [
    p for p in pps if p.getAttrs()["role"] == "exit"]
gStartnodes = sample(startnodes)
gGoalnodes = sample(goalnodes)

def proc0(sn, goal, waitTime, seatid):
    area = sn.getAttrs()["area"]
    generateAgents_(
        self.env.sampleInnerPathPoint(sn),
        None,
        None)
    a = self.agents[-1]
    a.area = area
    a.state = SFMStateWaiting(a)

    # 退出指示があるまで待つ。
    exitingAreasSeq = params["announce"]["area"]
    for k in range(len(exitingAreasSeq)):
        if area in exitingAreasSeq[k]:
            evtIndex = k
            break
    yield announceEvents[evtIndex].wait()

    yield pause(waitTime) # 指示を受けてからの待ち時間
    a.setDestination(goal, method = "CMM") # 移動を開始する。
```

エージェント集合の初期化処理 (2)

```
params = {
    "agent": {
        "a1": (0.80, goalnodes[1]),
        "a2": (0.80, goalnodes[1]),
        },
    "announce": {
        "area": (("a1",), ("a2",)),
        "threshold": (0.7, 1.0),
        },
    }

udist = uniformDistribution()
edist = exponentialDistribution(5)
for k, sn in enumerate(startnodes):
    area = sn.getAttrs()["area"]
    fillRate, goal = params["agent"][area]
    if next(udist) < fillRate:
        activate(proc0)(sn, goal, next(edist), k)
```

エージェント集合の初期化処理 (3)

```
announceEvents = [
    Event("-".join(a)) for a in params["announce"]["area"]]
def announce(param):
    exitingAreasSeq = param["area"]
    thresholdSeq = param["threshold"] # 退出率の閾値
    yield pause(1) # これがないと nExiting0 == 0 になる。
    for k in range(len(exitingAreasSeq)):
        # 退出指示を出す。
        yield announceEvents[k].signal()
        # 最初の exitingArea の人数を計算する。
        nExiting0 = len([
            a for a in self.agents
            if a.area in exitingAreasSeq[k]])
        # 5 秒ごとに現在の退出エリアの人の退出率を計算する
        # 毎秒計算するのは重いため
        while True:
            nExiting = len([
                a for a in self.agents
                if a.area in exitingAreasSeq[k]])
            nAgent = len([a for a in self.agents])
            try:
                exitRate = 1 - nExiting / nExiting0
            except ZeroDivisionError as ex:
                print("nExiting0 is zero; exitingAreas",
                      exitingAreasSeq[k])
                break
            yield pause(5)
            if exitRate > thresholdSeq[k]:
                break
activate(announce)(params["announce"])
```

次に、モデルのカスタムコード編集を開き、以下のコードを追加します。

カスタムコード編集

```
class SFMStateWaiting(sfm.SFMStateBase):
    pass
```

最後に、モデルメニューのパラメータを編集するを開き、シミュレーション終了時間の時間指定を 150 秒にします。(図 123)

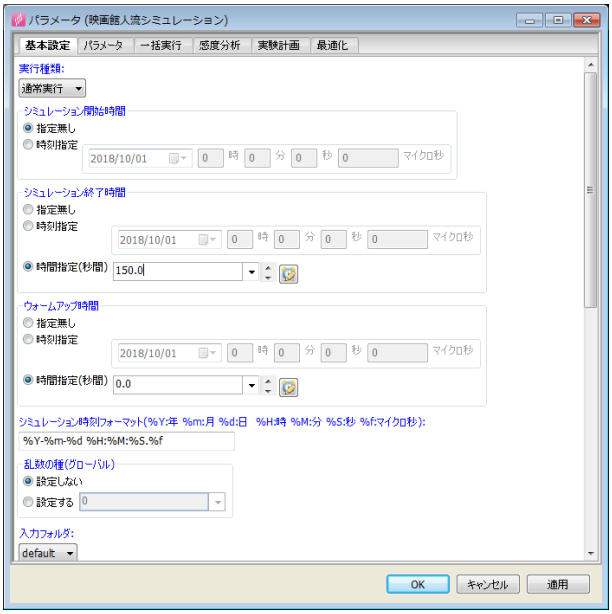


図 123: パラメータ編集

以上で設定は終了です。GUI のモデルメニューから「モデルを開始する」をクリックすると、シミュレーションが実行できます。(図 124)

11.13　指定領域内の人数を数える

地図エディタには、移動可能領域とは別に、任意の領域を指定してプログラム的に参照できるユーザー定義領域という機能があります。この機能を利用して、映画館のシミュレーションの各エリアに残っている観客の人数をリアルタイムグラフで表示する方法を説明します。

まず、地図エディタを更新します。SFM 地図アイコンの編集画面を開き、地図編集ボタンを押して地図エディタを開きます。ここでレイヤーを追加を選択してレイヤーを追加します。レイヤーはユーザー定義領域を格納できる概念で、モードとして選択できます。追加したレイヤーを選択します。

- 上側のエリアに対応するユーザー定義領域を作成します。多角形作成を選択し、客席全体を覆う形でユーザー定義領域を作成します。(図 125)
- 下側のエリアに対応するユーザー定義領域を作成します。矩形作成を選択し、客席全体を覆う形でユーザー定義領域を作成します。(図 126)

ユーザー定義領域を作成できましたら、属性を編集します。

- 追加したレイヤーの右クリックメニューから属性管理テーブルを表示を選択します。

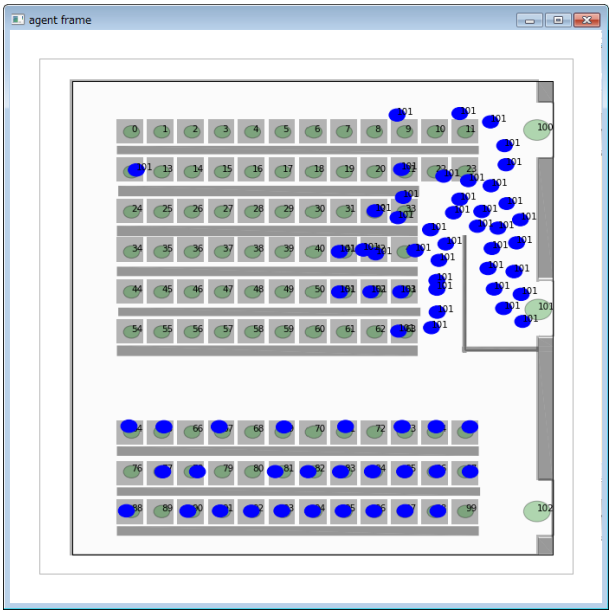


図 124: シミュレーション実行

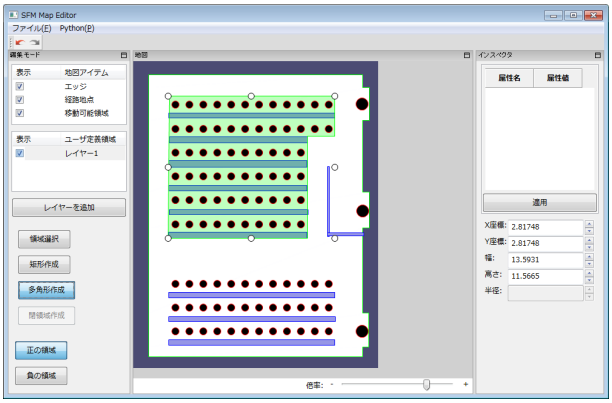


図 125: 上側のエリア

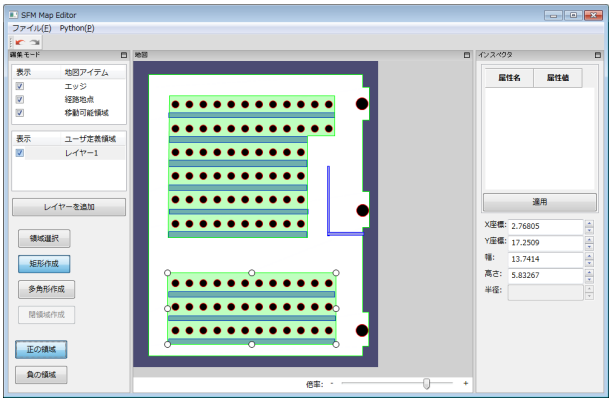


図 126: 下側のエリア

- 追加ボタンをクリックし、属性名に area を、デフォルト値に nothing を入力します。
- 属性管理テーブルを閉じます。
- 上側のエリアに対応するユーザー定義領域を選択し、右側のインスペクタで area の属性値を a1 にして、色が変わることを確認した上で適用ボタンをクリックします。
- 下側のエリアに対応するユーザー定義領域を選択し、右側のインスペクタで area の属性値を a2 にして、色が変わることを確認した上で適用ボタンをクリックします。

地図エディタの更新は以上です。閉じるボタンをクリックして保存して閉じます。

次に、エージェント部品を更新します。エージェント部品をダブルクリックして SFM エージェント編集画面を開き、エージェント集合の初期化処理を表示します。その冒頭に以下のコードを追加します。

エージェント集合の初期化処理

```
self.monitor = TimeMonitor(
    ["a1", "a2"], ["i", "i"], name = "領域内の人数")
self.simulator.addMonitor(self.monitor)
```

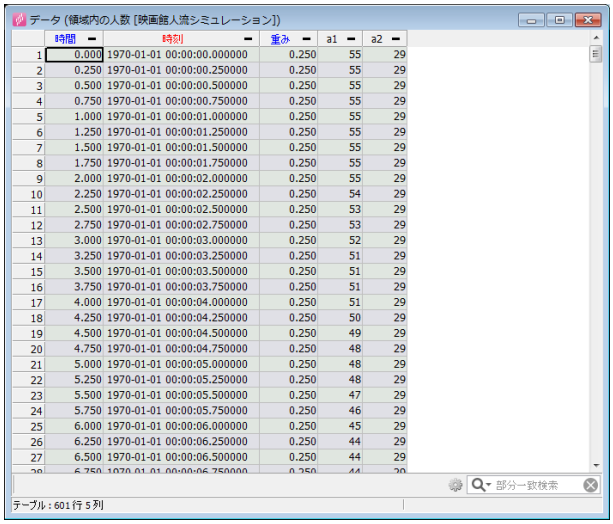
次に、SFM エージェントの編集画面からエージェント集合のステップ処理を表示します。その冒頭に以下のコードを追加します。

エージェント集合のステップ処理

```
lyr0 = self.env.getAllLayers()[0]
nagents = []
for area in ("a1", "a2"):
    uregion = [
        r for r in lyr0 if r.getAttrs()["area"] == area][0]
    n = len([
        a for a in self.agents
        if uregion.includes(*a.p.tolist())])
    nagents.append(n)
self.monitor.observe(now(), *nagents)
```

上記のコードが人数カウントの本質的な部分です。ユーザー定義領域 uregion が持つ includes メソッドでエージェントの位置 a.p が uregion 内にあるか否かを判定しています。

エージェント部品の更新は以上です。ここで一度シミュレーションを実行します。完了しましたらワークスペース出力の「領域内の人数」で右クリックメニューからデータを表示します。エリア a1, a2 それぞれの中にいる人数のデータが取れていることが確認できます。(図 127)



	時間	時刻	重み	a1	a2
1	0.000	1970-01-01 00:00:00.000000	0.250	55	29
2	0.250	1970-01-01 00:00:00.250000	0.250	55	29
3	0.500	1970-01-01 00:00:00.500000	0.250	55	29
4	0.750	1970-01-01 00:00:00.750000	0.250	55	29
5	1.000	1970-01-01 00:00:01.000000	0.250	55	29
6	1.250	1970-01-01 00:00:01.250000	0.250	55	29
7	1.500	1970-01-01 00:00:01.500000	0.250	55	29
8	1.750	1970-01-01 00:00:01.750000	0.250	55	29
9	2.000	1970-01-01 00:00:02.000000	0.250	55	29
10	2.250	1970-01-01 00:00:02.250000	0.250	54	29
11	2.500	1970-01-01 00:00:02.500000	0.250	53	29
12	2.750	1970-01-01 00:00:02.750000	0.250	53	29
13	3.000	1970-01-01 00:00:03.000000	0.250	52	29
14	3.250	1970-01-01 00:00:03.250000	0.250	51	29
15	3.500	1970-01-01 00:00:03.500000	0.250	51	29
16	3.750	1970-01-01 00:00:03.750000	0.250	51	29
17	4.000	1970-01-01 00:00:04.000000	0.250	51	29
18	4.250	1970-01-01 00:00:04.250000	0.250	50	29
19	4.500	1970-01-01 00:00:04.500000	0.250	49	29
20	4.750	1970-01-01 00:00:04.750000	0.250	48	29
21	5.000	1970-01-01 00:00:05.000000	0.250	48	29
22	5.250	1970-01-01 00:00:05.250000	0.250	48	29
23	5.500	1970-01-01 00:00:05.500000	0.250	47	29
24	5.750	1970-01-01 00:00:05.750000	0.250	46	29
25	6.000	1970-01-01 00:00:06.000000	0.250	45	29
26	6.250	1970-01-01 00:00:06.250000	0.250	44	29
27	6.500	1970-01-01 00:00:06.500000	0.250	44	29

図 127: データ表示

次に、リアルタイムグラフの設定を行います。ブラウザパネルで「グラフタブ」を選択してください。グラフタブの「リアルタイムグラフ」をモデル編集画面 (右側) にドラッグ&ドロップしてください。(図 128)

リアルタイムグラフをダブルクリックし、折れ線グラフをクリックして設定画面を開きます。ここで横軸を時間に、縦軸を a1, a2 に設定します。(図

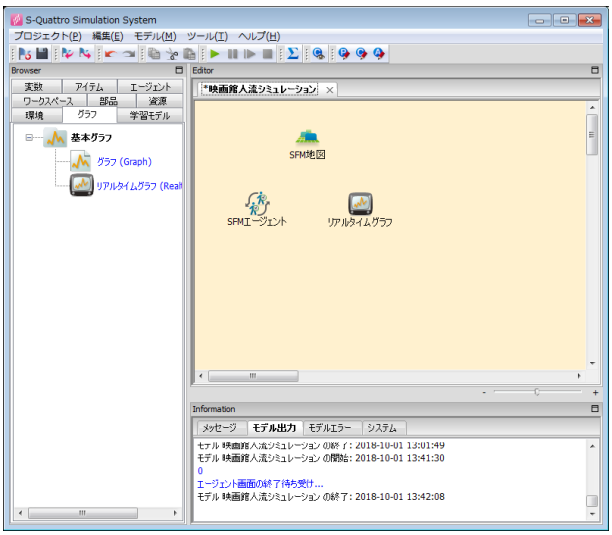


図 128: リアルタイムグラフの配置

129)

設定ができましたら再度シミュレーションを実行します。指定した領域内の人数がリアルタイムに表示されます。(図 130)

12　状態空間モデル

12.1　概要

状態空間モデルは、マルチエージェントシミュレーションのモデル化方法の一種です。S4 Simulation System では粒子フィルタによる状態空間モデル [1] が表現できます。ここでは、二次元空間内におけるオブジェクトの移動を例に説明します。

12.1.1　状態空間モデルについて

状態空間モデルは観測できない隠れた「状態」と、観測した結果である「観測値」からなるシミュレーションモデルです。具体的には次の二つの式で定義されます。

$$\begin{aligned} X_{t+1} &= f(X_t, u_t) \text{（状態方程式）} \\ Y_{t+1} &= g(X_{t+1}, v_{t+1}) \text{（観測方程式）} \end{aligned}$$

ここで、X_t は各時刻 $t = 1, 2, ..., T$ におけるシミュレーションモデルの状態、Y_t は各時刻における観測値、u_t はシステムノイズ、v_t は観測ノイズをあ

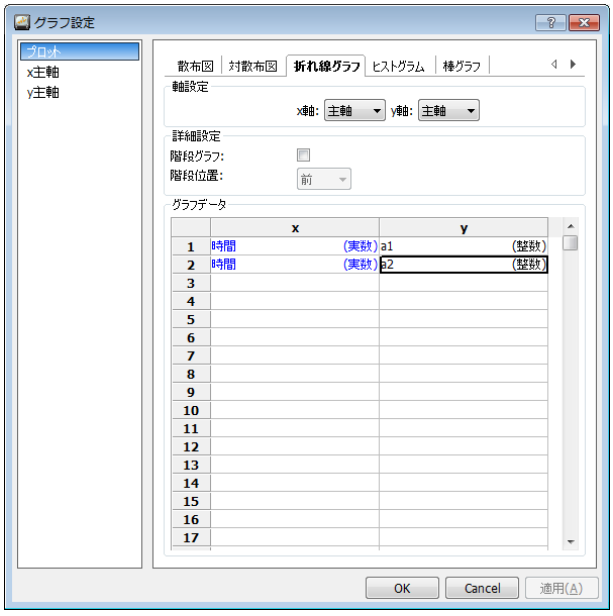


図 129: リアルタイムグラフの設定

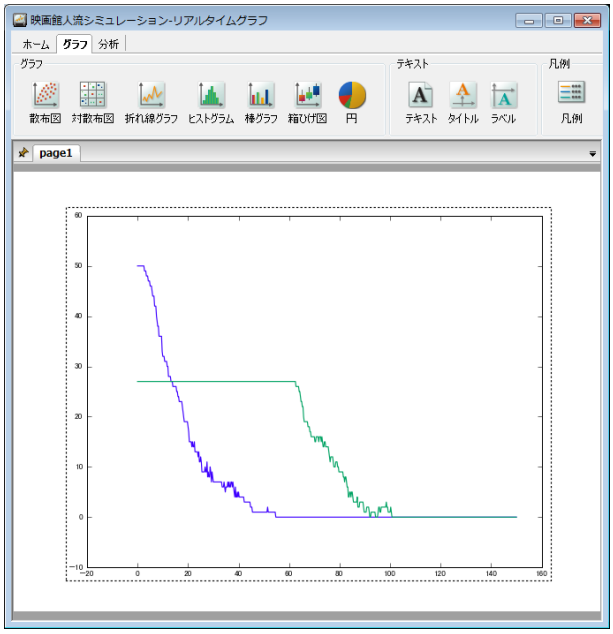


図 130: リアルタイムグラフ

らわします。また、f は状態遷移を司る関数、g は状態から観測値を生成する関数をあらわし、どちらの関数もノイズにより出力は確率的となります。

状態空間モデルによるモデル化は、他のマルチエージェントシミュレーションモデルと比べて、以下のような利点を持ちます。

データを用いたモデルのフィッティング・検証が行いやすい

シミュレーションモデルが状態空間モデルの形で表現されている場合、以下のような処理を繰り返すことにより、観測時系列データに当てはめながらシミュレーションを進められます。(図 131)

1. f を用いて現在の状態から次の状態を複数生成する。
2. 生成した次の状態と、観測値データを比較して、観測値データを生成しやすい状態を真の状態として選択する。

このようなシミュレーションを行うことで非決定性が強いモデルでも効率よく検証できます。

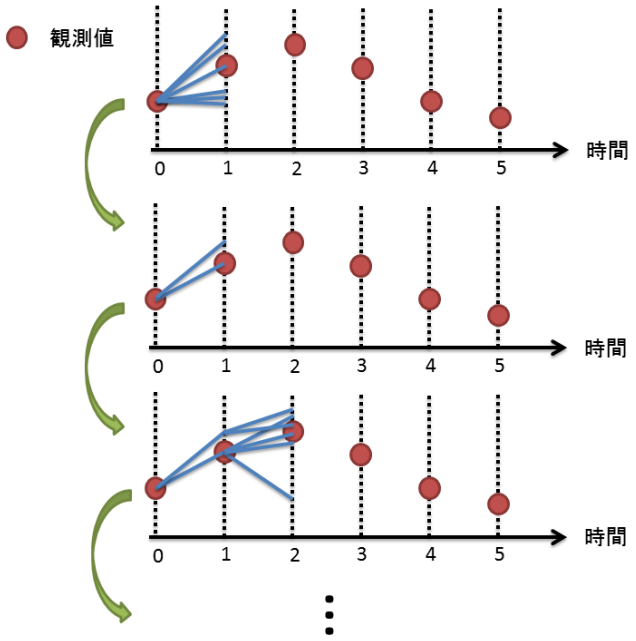


図 131: 状態空間モデル（粒子フィルタ）のイメージ

観測データの背後にあるモデルの（観測できない）内部状態が推定できる

状態空間モデルには状態と観測値が存在し、与えらたデータ（観測値）を生成するような状態を推測します。そのため、各エージェントの内部状態が推測できます。

Multi Target Tracking が可能となる

Multi Target Tracking (MTT) とは、観測値がエージェントごとに区別されていないデータに対して、観測値系列をエージェントと対応付けることをいいます。(図 132)

MTT は、各エージェントの挙動を状態空間モデルで表現し、エージェントの状態から生成されやすい観測値を対応付けることで実現されます。

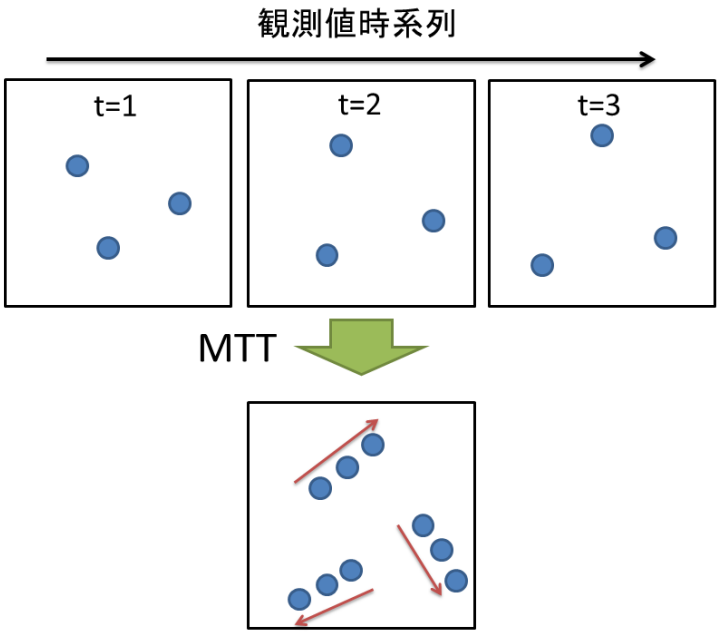


図 132: MTT のイメージ

12.1.2　本チュートリアルで作成するプロジェクトについて

本チュートリアルでは、二次元ユークリッド空間の中で与えられた観測時系列データに対して MTT を行い、複数のオブジェクトをトラッキングするプロジェクトを作成します。使用する状態空間モデルの状態方程式は以下の式で定義します。

$$
\begin{aligned}
x_{t+1} &= x_t + H(v_t, \theta_t) \\
H(v_t, \theta_t) &= \begin{pmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{pmatrix} v_t \\
v_{t+1} &= v_t + a_t + \sigma_v \eta \\
a_{t+1} &= a_t - \gamma(a_t - 0) + \sigma_a \rho \\
\theta_{t+1} &= \theta_t - \beta(\theta_t - 0) + \sigma_\theta \xi
\end{aligned}
$$

ここで、x_t は時刻 t における状態ベクトル、v_t、a_t、θ_t はそれぞれ時刻 t における速度ベクトル、加速度ベクトル、旋回角を表します。また、η、ρ、ξ は平均 0、分散 1 の正規分布に従うノイズ、σ_v、σ_a、σ_θ、β、γ はモデルのパラメータを表します。

観測方程式は以下の式で定義します。この式は、状態 x_t に Gaussian ノイズ u_t が加わったものを観測値とすることを意味し、尤度はガウス分布から定めます。

$$Y_{t+1} = x_{t+1} + u_{t+1}$$

12.2　状態空間モデルの作成手順

状態空間モデルを作成する際には、粒子フィルタエージェントを使用します。次節からは、粒子フィルタエージェントの設定方法を以下の順で説明します。

- プロジェクトの作成
- 環境の作成
- 入力データの用意
- エージェントの作成
- エージェントの環境・入力の設定
- 粒子の生成処理の設定
- 粒子の状態遷移の設定
- 粒子の尤度計算の設定
- 可視化処理の設定

12.3　プロジェクトの作成

S4 Simulation System を起動し、「プロジェクトメニュー」から「新規プロジェクト」を選択し、新規プロジェクトを作成します。「新規プロジェクトの作成」ダイアログが表示されますので、「状態空間モデル」と入力してください。ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されます。「モデル」をダブルクリックすると、モデル編集パネルに空のモデルが表示されます。

12.4　環境の作成

ブラウザパネルで「環境タブ」を選択してください。環境タブの「ユークリッド 2D」をドラッグし、モデル編集画面にドロップしてください。ユークリッド 2D が配置されます。

12.5　入力データの用意

状態空間モデルの作成や MTT を行う際に使用するデータは、一般に次のような形式が必要です。

時刻	観測値次元 1	観測値次元 2	⋯	観測値次元 n
0	$Y_1^{0,0}$	$Y_2^{0,0}$	⋯	$Y_n^{0,0}$
0	$Y_1^{0,1}$	$Y_2^{0,1}$	⋯	$Y_n^{0,1}$
0	$Y_1^{0,2}$	$Y_2^{0,2}$	⋯	$Y_n^{0,2}$
1	$Y_1^{1,0}$	$Y_2^{1,0}$	⋯	$Y_n^{1,0}$
⋮	⋮	⋮	⋯	⋮

データは、時刻をあらわす列とその時刻に観測された観測値をあらわす列で構成されます。データは時刻列について昇順に並んでおり、観測値は一定間隔ごとに観測されている必要があります。また、1 つの時刻に対して複数の行が存在してもよく、その場合は 1 行が 1 エージェントに対応します。

本チュートリアルで使用するデータは、次のような形式をもちます。

時刻	エージェント ID	x	y
0	0	$x_{0,0}$	$y_{0,0}$
0	1	$x_{0,1}$	$y_{0,1}$
0	2	$x_{0,2}$	$y_{0,2}$
1	0	$x_{1,0}$	$y_{1,0}$
⋮	⋮	⋮	⋮

各列の意味は以下の通りです。

時刻 観測値が記録された時刻をあらわす。観測値は一定時間ごとに観測されている。また、1 つの時刻に対して複数の行が存在する。その場合、同じ時刻をもつ行のエージェント ID は異なる。

エージェント ID 観測値を持つエージェントの ID である。可視化処理のみで使用し、MTT の処理本体では使用しない。

x, y 観測値である。エージェントが位置する座標をあらわす。

このようなデータを生成するプロジェクトをインポートし、既に生成してあるデータを「状態空間モデル」プロジェクトにコピーして使用します。プロジェクトのインポート、データのコピーは以下のようにして行います。

1. 「プロジェクトメニュー」から「プロジェクトをインポート」を選択し、S4 Simulation System のサンプルプロジェクトから「状態空間モデル用データ生成.s4」をインポートします。サンプルプロジェクトは、インストールフォルダ [1] 内の samples フォルダに用意されています。
2. ブラウザパネルで「ワークスペースタブ」を選択し、先ほどインポートした「状態空間モデル用データ生成」プロジェクトをダブルクリックします。その中の「出力」→ [default] フォルダの下にある「観測値の履歴」モニタを右クリックし、「データをコピーする」を選択します。
3. ブラウザパネル上にある「状態空間モデル」プロジェクトの「入力」→「default」フォルダを右クリックし、「データを貼り付ける」を選択します。

フィッティングするデータが csv 形式で用意されている場合でも、入力データとして使用できます。詳しくは、S4 Simulation System 操作マニュアル 4.4 節 「データのインポート」をご覧下さい。

12.6　エージェントの作成

ブラウザパネルで「エージェントタブ」を選択してください。 エージェントタブの「粒子フィルタエージェント」をドラッグし、モデル編集画面にドロップしてください。 粒子フィルタエージェントが配置されます。

12.7　エージェントの環境・入力の設定

「粒子フィルタエージェント」をダブルクリックすると編集画面が表示されます。(図 133)

まず、エージェントと環境を結び付けます。「環境オブジェクト」のプルダウンメニューで、「eEuclid2D(ユークリッド 2D)」を選択して下さい。

次に、先ほど用意した入力データを結びつけます。「フィッティングするデータ」の「モニタ名」のプルダウンメニューで、「観測値の履歴」を選択して

[1] インストールフォルダは、インストール時に特に指定しない場合
C:￥Program Files￥Mathematical Systems Inc￥S-Quattro Simulation System￥
となります。

下さい。また、時間列には「時刻」列が選択されていることを確認してください。

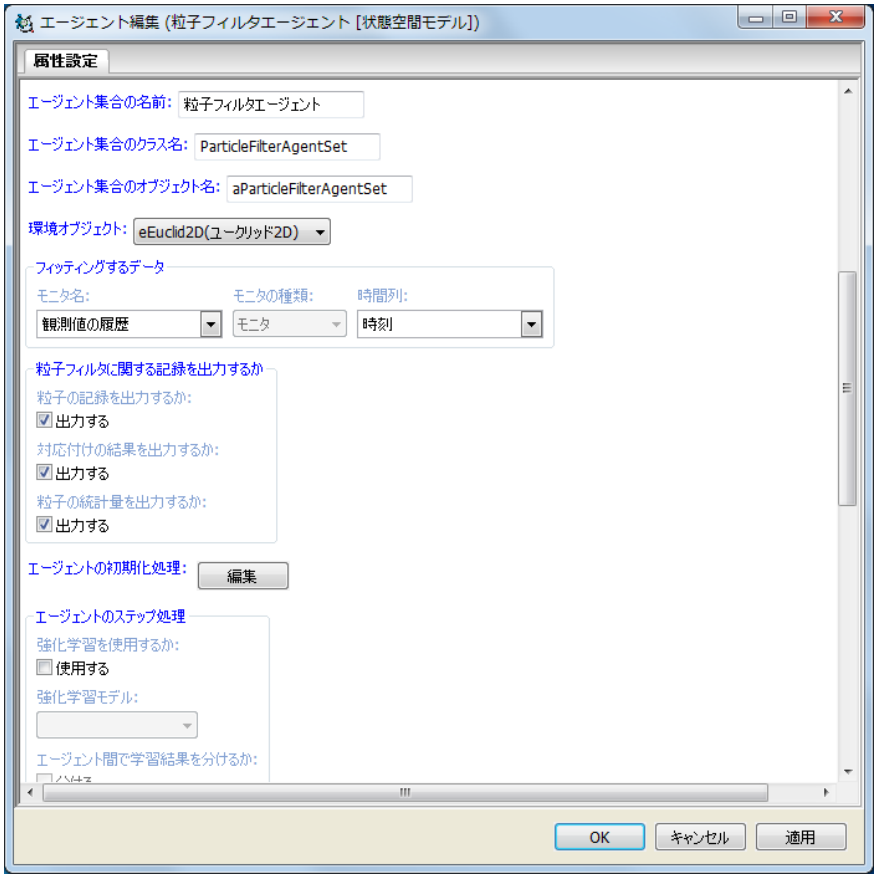


図 133: 粒子フィルタエージェント編集画面

12.8　粒子の生成処理の設定

「粒子フィルタに関する処理」内にある「粒子の生成処理」の「編集」をクリックします。すると、粒子の生成処理という名前のタブが開かれます。(図 134)

そのタブ内で、以下のコードを入力します。

粒子の生成処理

```
x0 = observation[0, u"x"]
y0 = observation[0, u"y"]
AgentParticles = self.agentset._defParticles()
n = 30     #生成する初期状態数（粒子数）
init_states = Monitor(cols=[u"x", u"y", u"vx", u"vy",
                            u"ax", u"ay", u"theta"])
for i in range(n):
    x = x0 + np.random.normal(loc=0.0, scale=0.01)
    y = y0 + np.random.normal(loc=0.0, scale=0.01)
    (vx, vy) = np.random.normal(loc=0.0, scale=0.01, size=2)
    (ax, ay) = np.random.normal(loc=0.0, scale=0.01, size=2)
    theta = np.random.normal(loc=0.0, scale=0.01)
    init_states.observe(x, y, vx, vy, ax, ay, theta)
return AgentParticles(init_states, observation)
```

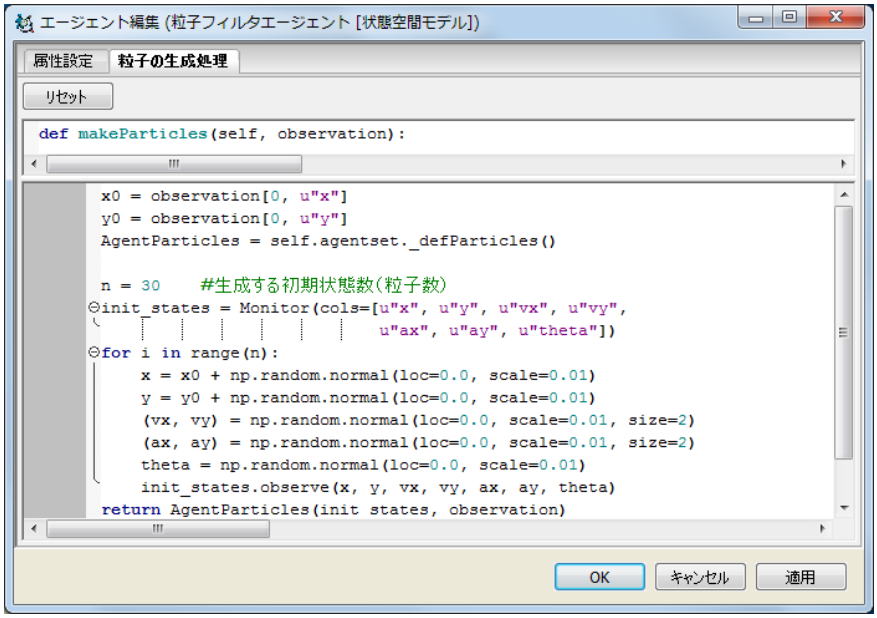


図 134: 粒子フィルタエージェント編集画面

「粒子の生成処理」では、観測値をもとにして初期状態を作成し、粒子群オブジェクトを生成します。

「AgentParticles = self.agentset._defParticles()」は、粒子フィルタに使用する粒子群クラスを取得します。

「n = 30」は、作成する初期状態の数をあらわします。

「init_states = Monitor(cols=[u"x", u"y", u"vx", ...」は、初期状態をあらわすモニタを定義します。

その後は与えられた観測値をもとにして、状態ベクトル、速度ベクトル、加速度ベクトル、旋回率といった状態の初期値を作成します。そして作成した状態と、作成に使用した観測値をもつ粒子群オブジェクトを返しています。

12.9　粒子の状態遷移の設定

粒子フィルタエージェントの「属性設定」タブに戻り、「粒子フィルタに関する処理」内にある「粒子の次の状態の計算処理」の「編集」をクリックします。すると、粒子の次の状態の計算処理という名前のタブが開かれます。そのタブ内で、以下のコードを入力します。

粒子の状態遷移の設定

```
#モデルパラメータの設定
sigma_v = 0.001
sigma_a = 0.001
sigma_theta = 0.001
gamma = 0.95
beta = 0.95
#ノイズの生成
n_row = states.nrow()
eta = np.random.randn(n_row, 2)
rho = np.random.randn(n_row, 2)
xi = np.random.randn(n_row, 1)
#状態の更新
next_states = states.copy()
states = np.array(states.toList()).T
(x, v, a, theta) = np.hsplit(states, [2,4,6])
for i in range(n_row):
    t_i = theta[i, 0]
    H_i = np.array([[np.cos(t_i), -np.sin(t_i)],
                    [np.sin(t_i),  np.cos(t_i)]])
    x[i,] += H_i.dot(v[i, :])
v += a + sigma_v * eta
a += -gamma * (a - 0) + sigma_a * rho
theta += - beta * (theta - 0) + sigma_theta * xi
next_states[:, :] = np.c_[x, v, a, theta].T.tolist()
return next_states
```

「粒子の次の状態の計算処理」では、状態方程式に基づいて現在の状態から次の状態を計算します。

最初に、状態方程式内に現れる各モデルパラメータを設定しています。その後は、状態方程式に従い次の状態を計算しています。

12.10　粒子の尤度計算の設定

粒子フィルタエージェントの「属性設定」タブに戻り、「粒子フィルタに関する処理」内にある「粒子の対数尤度計算処理」の「編集」をクリックします。すると、粒子の対数尤度計算処理という名前のタブが開かれます。そのタブ内で、以下のコードを入力します。

粒子の尤度計算の設定

```
from scipy.stats import norm
st = np.array([states[u"x"],
               states[u"y"]])
ob = np.array([observations[u"x"],
               observations[u"y"]])
errs = st[:,:,np.newaxis] - ob[:,np.newaxis,:]
likelihoods = norm.pdf(errs, scale=0.05)
loglikelihoods = np.sum(np.log(likelihoods), axis=0)
return loglikelihoods
```

「粒子の対数尤度計算処理」では、対応付けの際に使用する、状態と観測値の対数尤度を計算します。

「errs = st[:,:,np.newaxis] - ob[:,np.newaxis,:]」は、状態ベクトルと観測値の差を計算します。

「likelihoods = norm.pdf(errs, scale=0.05)」、「loglikelihoods = np.sum(np.log(likelihoods), axis=0)」は、計算した差から対数尤度を算出しています。観測方程式より、尤度はガウス分布で計算しています。

12.11　可視化処理の設定

可視化処理の設定は、次の 3 か所を編集します。

- 粒子フィルタエージェントの「エージェントの初期化処理」
- 粒子フィルタエージェントの「エージェント集合の可視化」
- ユークリッド 2D の「環境上のエージェントの描画処理」

「エージェントの初期化処理」では、各エージェントを初期化する際の処理を記述します。「エージェント集合の可視化」では、エージェント集合の可視化設定コードを記述します。また、「環境上のエージェントの描画処理」では、各エージェントの描画処理についてのコードを記述します。

まずは、粒子フィルタエージェントの「属性設定」タブに戻り、「エージェントの初期化処理」の編集をクリックし、以下のコードを入力します。

エージェントの初期化処理の設定

```
self.history = []
```

history は、各時刻における状態の平均値を保存しておくリストをあらわします。ここでは空のリストで初期化しています。

再び「属性設定」タブに戻り、「エージェント集合の可視化」の「編集」をクリックし、以下のコードを入力します。全てのコードを入力し終えたら、OK ボタンをクリックして編集画面を閉じます。

エージェント集合の可視化処理の設定

```
interval = 1 # 表示間隔
screen = self.getAgentScreen(interval = interval,
                             xlim = (-0.06, 1.06),
                             ylim = (-0.06, 1.06))
screen.addAgentSet(self)
screen.start()
```

次に、モデル編集パネル上にある「ユークリッド 2D」をダブルクリックします。するとユークリッド 2D の編集画面が表示されます。「環境上のエージェントの描画処理」の「編集」をクリックし、以下のコードを入力します。全てのコードを入力し終えたら、OK ボタンをクリックして編集画面を閉じます。

注意: 環境上のエージェントの描画処理の設定 (1) と環境上のエージェントの描画処理の設定 (2) に分かれていますが、両方を「環境上のエージェントの描画処理」内につなげて入力します。

環境上のエージェントの描画処理の設定 (1)

```
from scipy.stats import norm
screen = panel.screen
isVisualizeParticles = True

for agent in agents:
    if agent.particles.observation is None:
        continue
    particles = agent.particles
    if isVisualizeParticles:
        for i in range(particles.states.nrow()):
            sx = particles.states[i,u"x"]
            sy = particles.states[i,u"y"]
            #粒子の色設定
            (max_val, min_val) = (65, 0)
            width = max_val - min_val
            br = np.exp(particles.loglikelihoods[i]) / width
            r = int((0xFF - 0x55 + 0x00) * (1-br) + 0x00)
            g = int((0xFF - 0x55 + 0x00) * (1-br) + 0x55)
            b = int((0xFF - 0x55 + 0x00) * (1-br) + 0x00)
            color_code = "#%02x%02x%02x"%(r, g, b)
            #粒子の描画
            screen.point(sx, sy, size = 30,
                         color = color_code,
                         marker = 'o',
                         alpha = 0.20)
    (mx, my) = particles.calcStats("mean")[0:2]
    #平均の描画
    screen.point(mx, my, size = 95,
                 color = 'b',
                 marker = 'o',
                 alpha = 1.0)
    screen.text(mx, my+0.01, str(agent.agentid),
                fontsize = 15,
                color = 'b')
```

環境上のエージェントの描画処理の設定 (2)

```
#観測値の描画
x = particles.observation[0,u"x"]
y = particles.observation[0,u"y"]
truth_id = particles.observation[0,u"エージェント ID"]
screen.point(x, y, size = 95,
             color = 'r',
             marker = 'o',
             alpha = 0.7)
screen.text(x, y+0.01, str(truth_id),
            fontsize = 15,
            color = 'k')
screen.lines([(x,y),(mx,my)],
             linewidth = 1.5,
             color = 'b',
             linestyle = "solid",
             cmap = None,
             alpha = 1.0)
#95%境界線の描画
r = norm.ppf(q=0.95, loc=0, scale=0.05)
screen.ellipse(mx, my,
               width=2*r, height=2*r,
               color="w", edgecolor='b',
               linewidth=1.0, zorder=-1, alpha=0.3)
#軌跡の描画
agent.history.append((mx, my))
screen.lines(agent.history,
             linewidth = 1.0,
             color = 'k',
             linestyle = "dashed",
             cmap = None,
             alpha = 1.0)
```

「isVisualizeParticles = True」は、エージェントの状態を描画するかどうかを設定します。False を選択すると、状態に関しては平均座標のみを描画します。

「screen.point(sx, sy, size = 20, ...」は、状態の座標にマーカーを設置します。マーカーの色は緑色とし、尤度が大きいほど濃くなります。

「screen.point(mx, my, size = 50, ...」は、状態の平均の位置にマーカーを設置します。マーカーの色は青色としています。

「screen.point(x, y, size = 50, ...」は、観測値の座標にマーカーを設置します。マーカーの色は赤色としています。

「screen.ellipse(mx, my, width=2*r, height=2*r, ...」は、平均座標を中心として、ノイズが等方的な正規分布であると仮定した場合の 95 値%境界を描画します。

「screen.lines(agent.history, ...」は、状態の平均の軌跡を表示します。エージェントの history 属性に保存された各座標を結ぶ直線を引くことで描画されます。

12.12　パラメータの設定

実行する前にモデル全体のパラメータを設定します。「モデルメニュー」から「パラメータを編集する」を開き、シミュレーション終了時間を 100 に設定します。また、今回は乱数の種を 0 に設定します。

12.13　モデルの実行

モデルメニューから「モデルを開始する」を選ぶと、モデルが実行されます。以下のようなエージェントフレームが表示されます。(図 135)

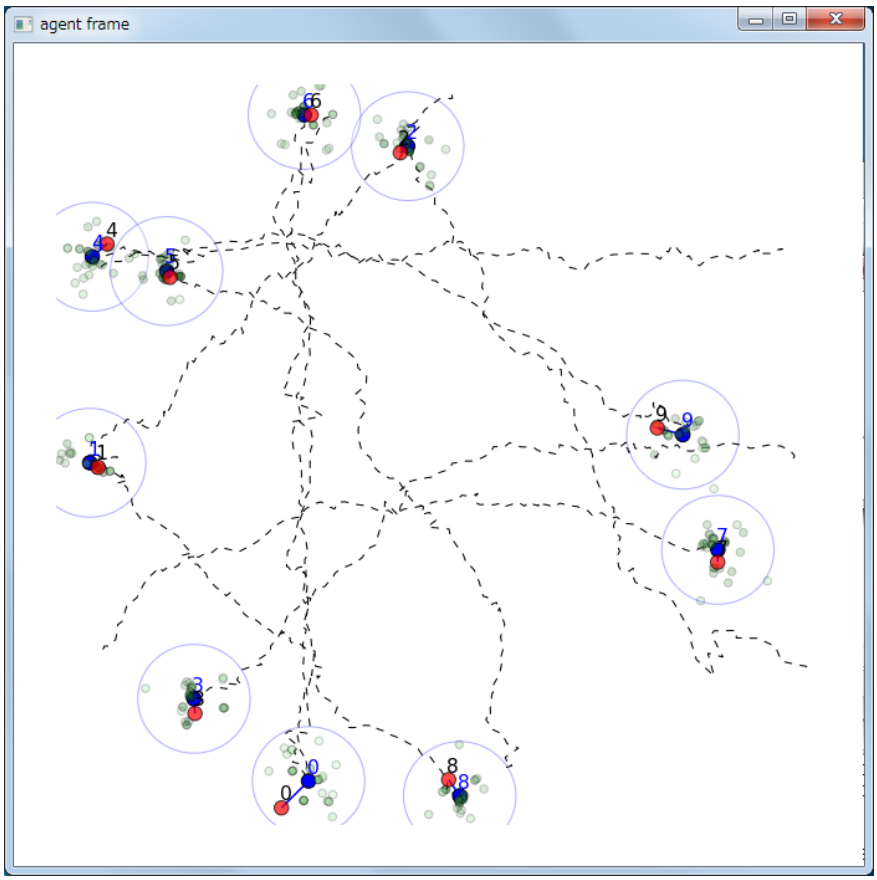


図 135: 観測値データに基づくシミュレーション結果

青色の点が状態の平均座標、赤色の点が観測値をあらわし、表示されている数字はエージェント ID をあらわします。青色の直線は、エージェントと観測値の対応付け結果をあらわしたもので、同じ ID のエージェントと観測値が結ばれていた場合、正しい対応付けが行われたことを意味します。緑色の点は状態を表し、尤度が高いほど色が濃くなっています。また、青色の円は 95%値境界を表しています。

モデルの実行が終了しエージェントフレームを閉じると、ブラウザパネル上にある「状態空間モデル」プロジェクトの「出力」→「default」フォルダに、以下の 3 つのモニタが出力されます。

- 対応付けの結果
- 粒子の記録
- 粒子の統計量

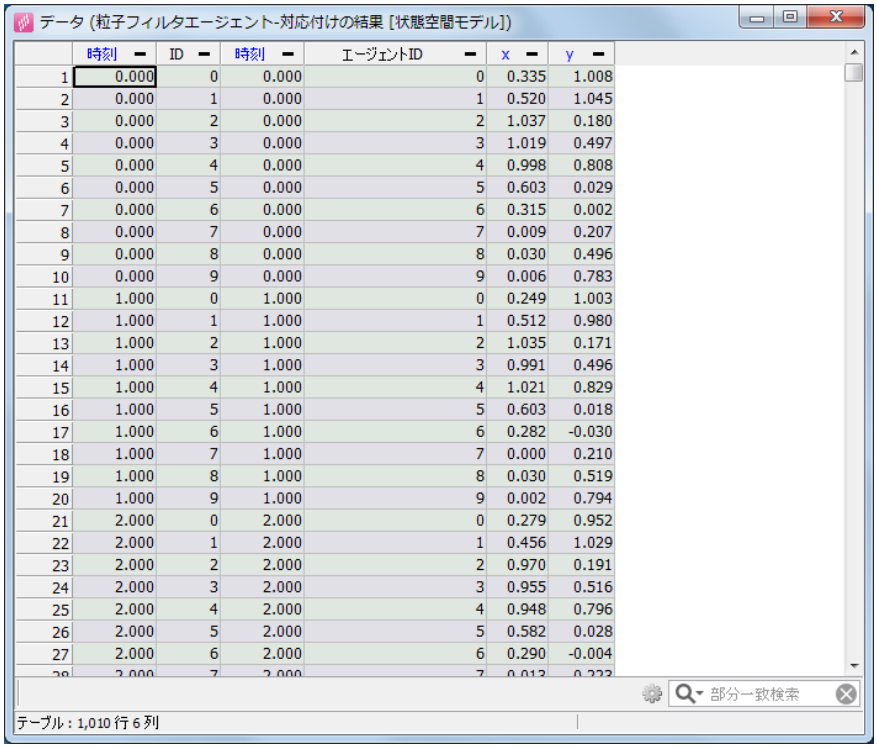
それぞれのデータは、モニタ名を右クリックし、「データを表示する」を選択すると閲覧できます。

対応付けの結果 (図 136) は、MTT によるエージェントと観測値の対応付け結果を時刻ごとに記録したモニタであり、入力で与えたデータの左側に次の 2 列が挿入された形式をもちます。

時刻 シミュレーション時刻をあらわします。

ID MTT により同じ行の観測値に対応付けられたエージェントの ID を表します。

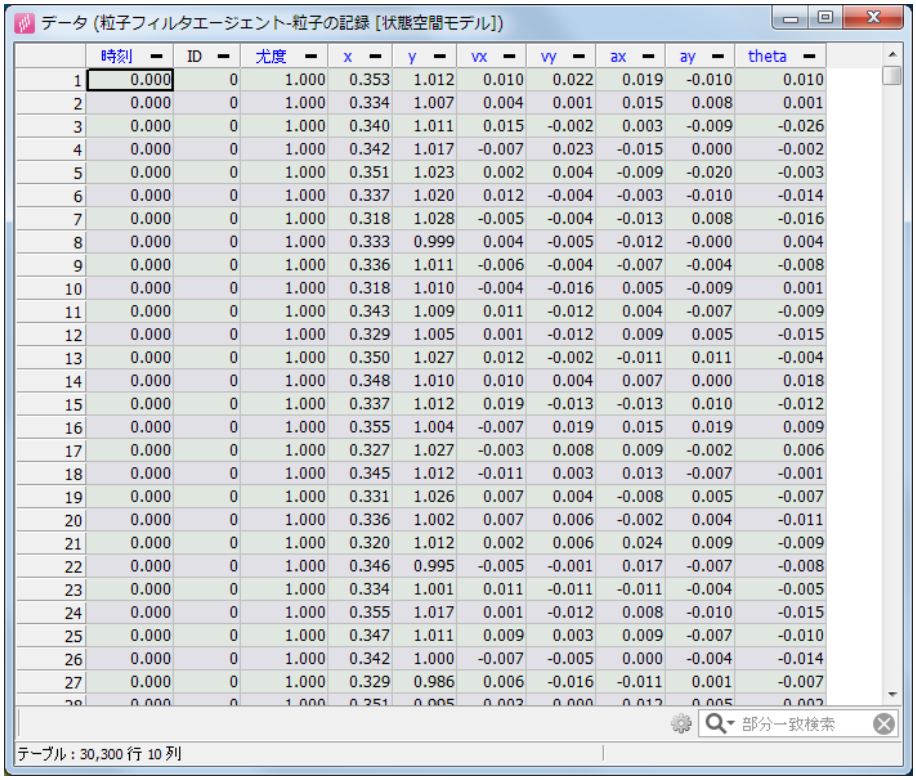
今回のプロジェクトでは、この「ID」列と入力データの「エージェント ID」列の値が等しければ、正しい対応付けが行われています。



	時刻	ID	時刻	エージェントID	x	y
1	0.000	0	0.000	0	0.335	1.008
2	0.000	1	0.000	1	0.520	1.045
3	0.000	2	0.000	2	1.037	0.180
4	0.000	3	0.000	3	1.019	0.497
5	0.000	4	0.000	4	0.998	0.808
6	0.000	5	0.000	5	0.603	0.029
7	0.000	6	0.000	6	0.315	0.002
8	0.000	7	0.000	7	0.009	0.207
9	0.000	8	0.000	8	0.030	0.496
10	0.000	9	0.000	9	0.006	0.783
11	1.000	0	1.000	0	0.249	1.003
12	1.000	1	1.000	1	0.512	0.980
13	1.000	2	1.000	2	1.035	0.171
14	1.000	3	1.000	3	0.991	0.496
15	1.000	4	1.000	4	1.021	0.829
16	1.000	5	1.000	5	0.603	0.018
17	1.000	6	1.000	6	0.282	-0.030
18	1.000	7	1.000	7	0.000	0.210
19	1.000	8	1.000	8	0.030	0.519
20	1.000	9	1.000	9	0.002	0.794
21	2.000	0	2.000	0	0.279	0.952
22	2.000	1	2.000	1	0.456	1.029
23	2.000	2	2.000	2	0.970	0.191
24	2.000	3	2.000	3	0.955	0.516
25	2.000	4	2.000	4	0.948	0.796
26	2.000	5	2.000	5	0.582	0.028
27	2.000	6	2.000	6	0.290	-0.004

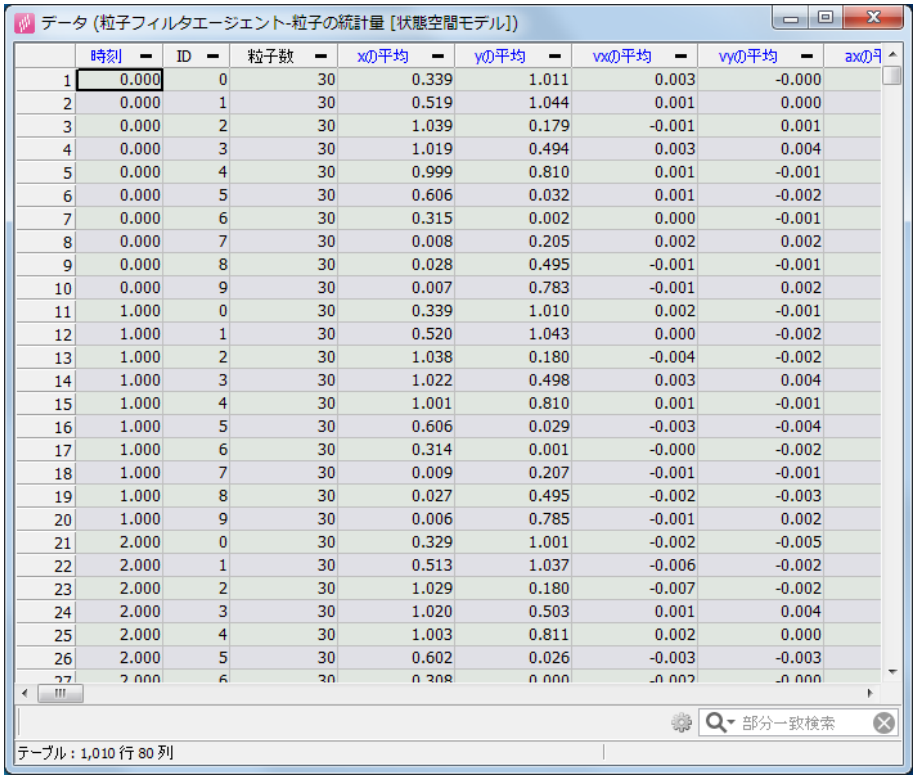
図 136: 対応付けの結果

粒子の記録 (図 137)・粒子の統計量 (図 138) は、モデル実行中に生成されたエージェントの状態とその統計量が時刻ごとに記録されています。



	時刻	ID	尤度	x	y	vx	vy	ax	ay	theta
1	0.000	0	1.000	0.353	1.012	0.010	0.022	0.019	-0.010	0.010
2	0.000	0	1.000	0.334	1.007	0.004	0.001	0.015	0.008	0.001
3	0.000	0	1.000	0.340	1.011	0.015	-0.002	0.003	-0.009	-0.026
4	0.000	0	1.000	0.342	1.017	-0.007	0.023	-0.015	0.000	-0.002
5	0.000	0	1.000	0.351	1.023	0.002	0.004	-0.009	-0.020	-0.003
6	0.000	0	1.000	0.337	1.020	0.012	-0.004	-0.003	-0.010	-0.014
7	0.000	0	1.000	0.318	1.028	-0.005	-0.004	-0.013	0.008	-0.016
8	0.000	0	1.000	0.333	0.999	0.004	-0.005	-0.012	-0.000	0.004
9	0.000	0	1.000	0.336	1.011	-0.006	-0.004	-0.007	-0.004	-0.008
10	0.000	0	1.000	0.318	1.010	-0.004	-0.016	0.005	-0.009	0.001
11	0.000	0	1.000	0.343	1.009	0.011	-0.012	0.004	-0.007	-0.009
12	0.000	0	1.000	0.329	1.005	0.001	-0.012	0.009	0.005	-0.015
13	0.000	0	1.000	0.350	1.027	0.012	-0.002	-0.011	0.011	-0.004
14	0.000	0	1.000	0.348	1.010	0.010	0.004	0.007	0.000	0.018
15	0.000	0	1.000	0.337	1.012	0.019	-0.013	-0.013	0.010	-0.012
16	0.000	0	1.000	0.355	1.004	-0.007	0.019	0.015	0.019	0.009
17	0.000	0	1.000	0.327	1.027	-0.003	0.008	0.009	-0.002	0.006
18	0.000	0	1.000	0.345	1.012	-0.011	0.003	0.013	-0.007	-0.001
19	0.000	0	1.000	0.331	1.026	0.007	0.004	-0.008	0.005	-0.007
20	0.000	0	1.000	0.336	1.002	0.007	0.006	-0.002	0.004	-0.011
21	0.000	0	1.000	0.320	1.012	0.002	0.006	0.024	0.009	-0.009
22	0.000	0	1.000	0.346	0.995	-0.005	-0.001	0.017	-0.007	-0.008
23	0.000	0	1.000	0.334	1.001	0.011	-0.011	-0.011	-0.004	-0.005
24	0.000	0	1.000	0.355	1.017	0.001	-0.012	0.008	-0.010	-0.015
25	0.000	0	1.000	0.347	1.011	0.009	0.003	0.009	-0.007	-0.010
26	0.000	0	1.000	0.342	1.000	-0.007	-0.005	0.000	-0.004	-0.014
27	0.000	0	1.000	0.329	0.986	0.006	-0.016	-0.011	0.001	-0.007

図 137: 粒子の記録



	時刻	ID	粒子数	xの平均	yの平均	vxの平均	vyの平均	axの平
1	0.000	0	30	0.339	1.011	0.003	-0.000	
2	0.000	1	30	0.519	1.044	0.001	0.000	
3	0.000	2	30	1.039	0.179	-0.001	0.001	
4	0.000	3	30	1.019	0.494	0.003	0.004	
5	0.000	4	30	0.999	0.810	0.001	-0.001	
6	0.000	5	30	0.606	0.032	0.001	-0.002	
7	0.000	6	30	0.315	0.002	0.000	-0.001	
8	0.000	7	30	0.008	0.205	0.002	0.002	
9	0.000	8	30	0.028	0.495	-0.001	-0.001	
10	0.000	9	30	0.007	0.783	-0.001	0.002	
11	1.000	0	30	0.339	1.010	0.002	-0.001	
12	1.000	1	30	0.520	1.043	0.000	-0.002	
13	1.000	2	30	1.038	0.180	-0.004	-0.002	
14	1.000	3	30	1.022	0.498	0.003	0.004	
15	1.000	4	30	1.001	0.810	0.001	-0.001	
16	1.000	5	30	0.606	0.029	-0.003	-0.004	
17	1.000	6	30	0.314	0.001	-0.000	-0.002	
18	1.000	7	30	0.009	0.207	-0.001	-0.001	
19	1.000	8	30	0.027	0.495	-0.002	-0.003	
20	1.000	9	30	0.006	0.785	-0.001	0.002	
21	2.000	0	30	0.329	1.001	-0.002	-0.005	
22	2.000	1	30	0.513	1.037	-0.006	-0.002	
23	2.000	2	30	1.029	0.180	-0.007	-0.002	
24	2.000	3	30	1.020	0.503	0.001	0.004	
25	2.000	4	30	1.003	0.811	0.002	0.000	
26	2.000	5	30	0.602	0.026	-0.003	-0.003	
27	2.000	6	30	0.308	0.000	-0.002	-0.000	

図 138: 粒子の統計量

次に、実行結果を確認するため、入力データとの比較を行います。「ワークスペースタブ」から 12.5 節でインポートした「状態空間モデル用データ生成」プロジェクトの「モデル」をダブルクリックして開きます。次に、モデルメニューから「モデルを開始する」を選択すると、以下のようなエージェントフレームが表示されます。(図 139)

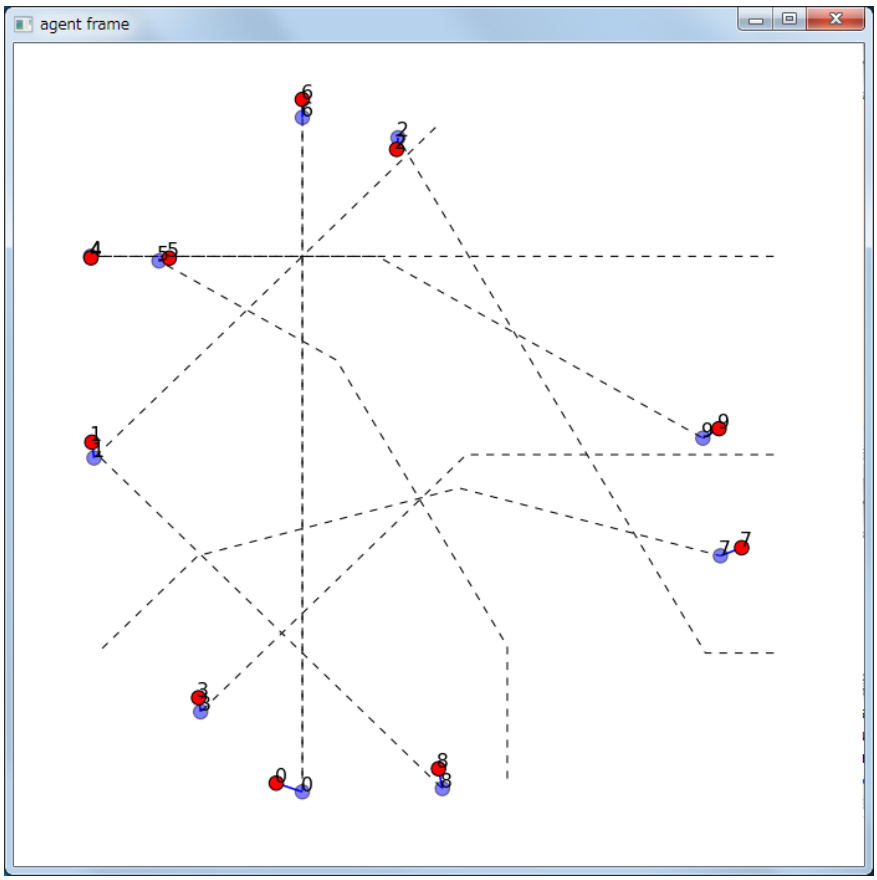


図 139: 観測値データに基づくシミュレーション結果

青色の点が各エージェントの内部状態、赤色の点が状態から得られた観測値をあらわし、表示されている数字は ID をあらわします。この実行結果をみると、状態空間モデルを使用したシミュレーション (図 135) は与えられた観測値データから内部状態をよく再現していることが分かります。

12.14　モデルパラメータの最適化

作成した状態空間モデルは、状態方程式に含まるモデルパラメータを、「粒子フィルタエージェント」の「粒子の次の状態の計算処理」内で指定していました。該当する部分のみを抜き出すと、次のコードになります。

モデルパラメータの設定

```
sigma_v = 0.001
sigma_a = 0.001
sigma_theta = 0.001
gamma = 0.95
beta = 0.95
```

各パラメータに代入する値は、本来は何回もモデルを実行して適切な値を見つける必要があります。このパラメータの調節が手動では難しい場合、S4 Simulation System に含まれる最適化機能を使用してパラメータを自動で調節できます。

最適化では、最適化すべき目的関数とその際のパラメータを指定します。今回の例では、モデル全体の対数尤度を目的関数とし、モデルの状態方程式に含まれる σ_v、σ_a、σ_θ、γ、β をパラメータとします。そして、目的関数を最大化するパラメータを最適化によって求めます。

最適化処理の設定は、次の手順で行います。

- パラメータの設定
- 粒子フィルタエージェントの設定
- 目的関数の設定
- 最適化設定

12.14.1　パラメータの設定

状態空間モデルにおける最適化のためのパラメータを編集します。モデル編集パネル上に「状態空間モデル」プロジェクトが表示されていることを確認して、「モデル」メニューの「パラメータを編集する」を選択します。

あらわれたパラメータ編集画面で「パラメータ」タブを選択します。ここでは、シミュレーションで用いる変数を定義します。(図 140)

「パラメータ」内にある「+」ボタンを押してパラメータを追加します。 変数名、生成方式、型、値は以下のように設定します。全てのパラメータの設定が終了したら、「OK」ボタンをクリックして編集画面を閉じます。

変数名	生成方式	型	値
sigma_v	固定	実数	0.001
sigma_a	固定	実数	0.001
sigma_theta	固定	実数	0.001
gamma	固定	実数	0.95
beta	固定	実数	0.95

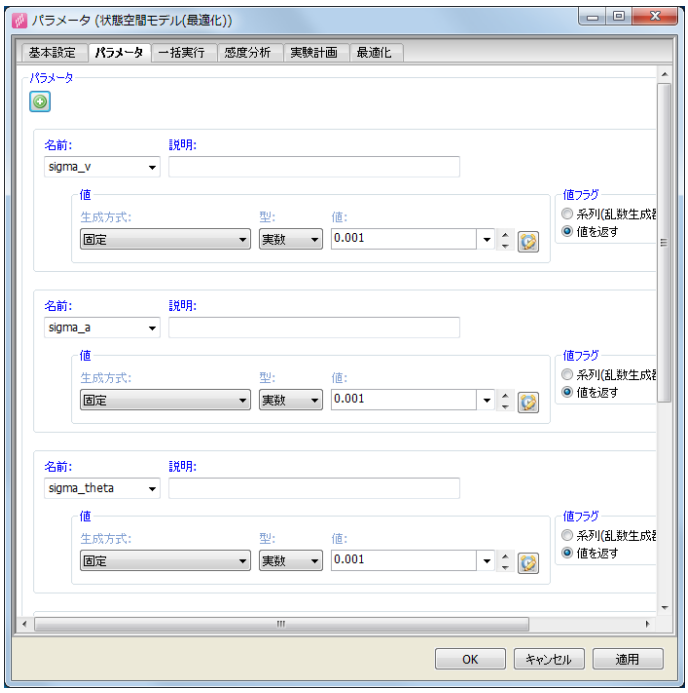


図 140: パラメータ 編集画面

12.14.2　粒子フィルタエージェントの設定

次に、粒子フィルタエージェントをダブルクリックして編集画面を開きます。その中にある「粒子の次の状態の計算処理」の「編集」ボタンをクリックして編集画面を開きます。(図 141)

モデルパラメータの設定部分を以下のコードに書き換えます。

モデルパラメータの設定

```
sigma_v = param.sigma_v
sigma_a = param.sigma_a
sigma_theta = param.sigma_theta
gamma = param.gamma
beta = param.beta
```

これにより、実行時にこれらのパラメータを用いてシミュレーションが行われるようになります。編集が終了したら「OK」ボタンをクリックして編集画面を閉じます。

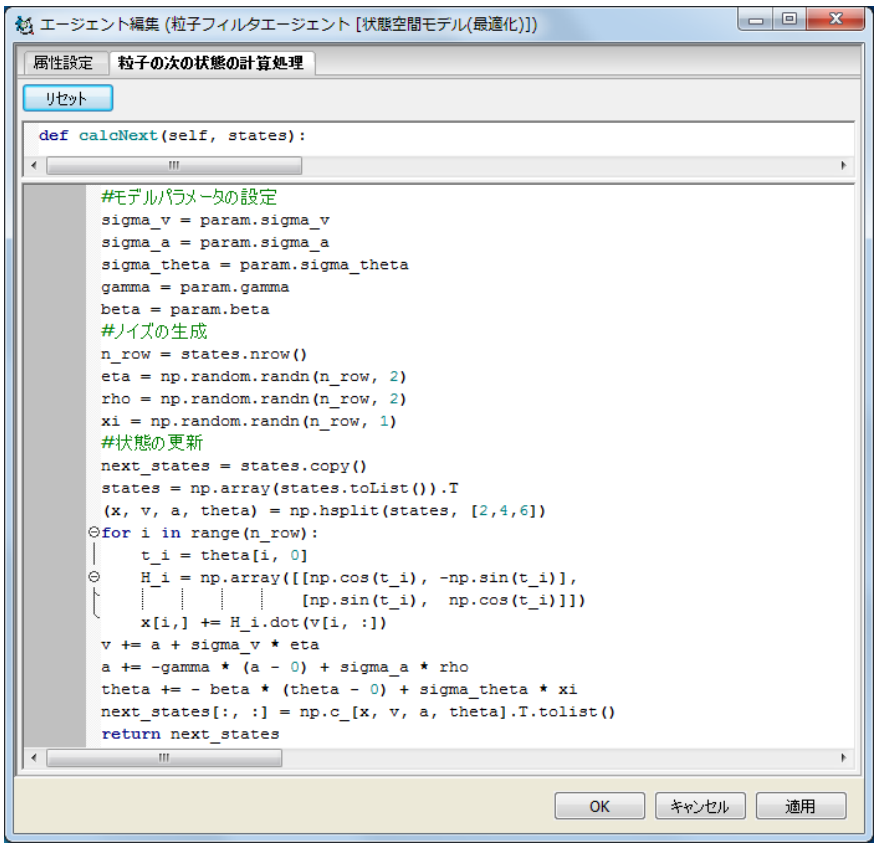


図 141: 粒子の次の状態の計算処理の設定

12.14.3　目的関数の設定

最適化の目的関数を編集します。「モデル」メニューの「分析スクリプトを編集する」を選択し、分析スクリプト編集画面を開きます。(図 142)

目的関数がモデル全体の対数尤度となるように、objective の定義を以下のコードに書き換えます。編集が終了したら「OK」ボタンをクリックして編集画面を閉じます。

目的関数の設定

```
return self.aParticleFilterAgentSet.filter.loglikelihood
```

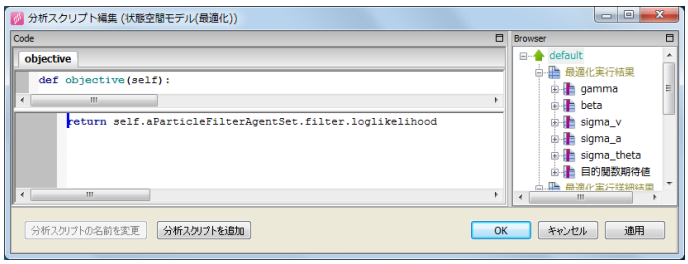


図 142: 分析スクリプト 編集画面

12.14.4　最適化設定

実行のオプションとして最適化実行を行う設定をします。 「モデル」メニューの「パラメータを編集する」を選択し、パラメータ編集画面を再び開きます。

まず、「基本設定」タブの「実行種類」を「最適化」にします。

次に、「最適化」タブで最適化に用いるパラメータを定義します。「最適化設定」内にある「+」ボタンを押して最適化変数を追加します。 パラメータ名には先ほど設定したパラメータをコンボリストから選択し、以下のように設定します。(図 143)

パラメータ名	型	下限	上限
sigma_v	実数	0	0.01
sigma_a	実数	0	0.01
sigma_theta	実数	0	0.01
gamma	実数	0.5	1
beta	実数	0.5	1

最適化のオプションは以下のように指定します。(図 144)

目的関数はデフォルトの self.objective のままで構いません。また、最大化/最小化の欄もデフォルトの最大化を指定します。

シミュレーションの結果は確率的に変わるため、目的関数の値は一意に求めることができません。 そこで、シミュレーションの最適化においては、目的関数の期待値を計算します。期待値を計算するためのサンプル数がレプリケーションとなります。ここでは最大回数を指定することもできますし、目的関数の信頼区間で指定することもできます。 今回は最小回数を 1、最大回数を 5 で行います。

最適化は Particle Swarm Optimization(PSO) と呼ばれる手法を用いて最適化を行います。最適化オプションではこれらのパラメータを設定します。今回のチュートリアルではデフォルトのまま行います。 この状態で「OK」ボタンを押し編集画面を閉じます。

図 143: 最適化パラメータ 編集画面

図 144: 最適化パラメータ 編集画面 2

12.14.5　最適化実行

「モデルメニュー」から「モデルを開始する」を選択し、シミュレーション最適化を開始します。最適化を実行するとブラウザパネル上にある「状態空間モデル」プロジェクトの「出力」→「default」フォルダに、「最適化実行結果」と「最適化実行詳細結果」の 2 つのモニタが出力されます。

「最適化実行結果」の上で右クリックをし、「データを表示する」を選択すると以下のようなモニタが表示されます。(図 145) 「目的関数期待値」と書かれた列にある右側の「—」を 2 回クリックすると目的関数期待値について降順に並べ替えることができます。並べ替えた後に 1 行目にある数値が、最適化によって求められた対数尤度を最大化する各パラメータの値になります。このように最適化機能を使用することによって、状態空間モデルのモデルパラメータを自動的に決定することができます。

	sigma_v	sigma_a	sigma_theta	gamma	beta	1. 目的関数期待値
1	0.002	0.000	0.001	1.000	1.000	3659.716
2	0.002	0.000	0.001	1.000	1.000	3655.037
3	0.002	0.000	0.001	1.000	1.000	3650.674
4	0.002	0.000	0.001	1.000	1.000	3650.313
5	0.002	0.000	0.001	1.000	1.000	3649.525
6	0.002	0.000	0.001	1.000	1.000	3649.339
7	0.002	0.000	0.000	0.791	0.706	3649.309
8	0.002	0.000	0.002	1.000	1.000	3648.911
9	0.002	0.000	0.002	1.000	1.000	3648.650
10	0.002	0.000	0.004	1.000	1.000	3648.281
11	0.002	0.000	0.000	1.000	1.000	3647.839
12	0.002	0.000	0.001	1.000	1.000	3647.393
13	0.002	0.000	0.000	1.000	1.000	3647.141
14	0.002	0.000	0.000	1.000	1.000	3646.970
15	0.002	0.000	0.000	1.000	1.000	3646.883
16	0.002	0.000	0.004	1.000	1.000	3646.841
17	0.002	0.000	0.000	1.000	1.000	3646.464
18	0.002	0.000	0.002	1.000	0.978	3646.180
19	0.002	0.000	0.000	1.000	0.997	3645.675
20	0.002	0.000	0.001	1.000	1.000	3645.575
21	0.002	0.000	0.001	1.000	1.000	3644.918
22	0.002	0.000	0.001	0.996	0.983	3644.681
23	0.002	0.000	0.001	1.000	1.000	3643.923
24	0.002	0.000	0.000	1.000	1.000	3643.754
25	0.002	0.000	0.001	1.000	1.000	3643.294
26	0.002	0.000	0.002	1.000	1.000	3642.778
27	0.002	0.000	0.000	1.000	1.000	3642.768

図 145: 最適化実行結果

13　窓口モデル用部品

S4 Simulation System は汎用ツールですが、より簡単にモデルが作れるように特定のモデルに特化した部品もいくつか提供しています。2 章の銀行の

窓口モデルでは、アイテムやファシリティといった汎用部品を使ってモデル化をしましたが、ここでは窓口モデル用の部品を使ったモデルを見ていきます。ここで紹介するモデルはテンプレートモデルとしてサンプルに含まれています。

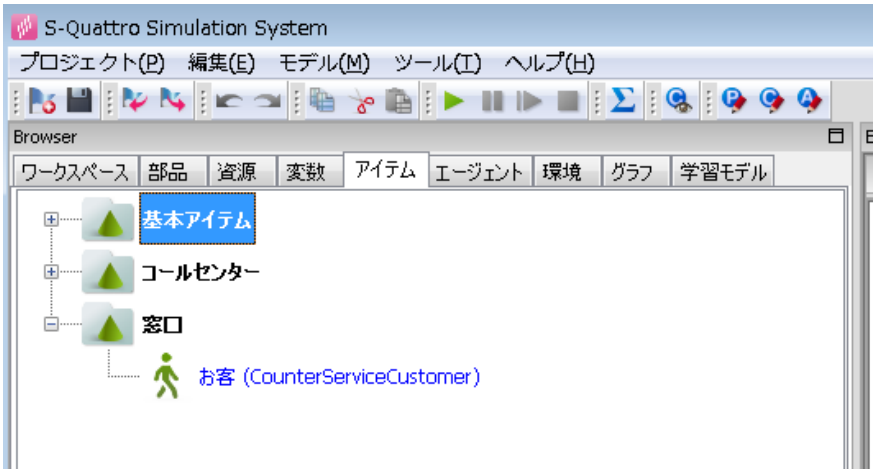


図 146: 窓口モデル用部品（アイテムタブ）

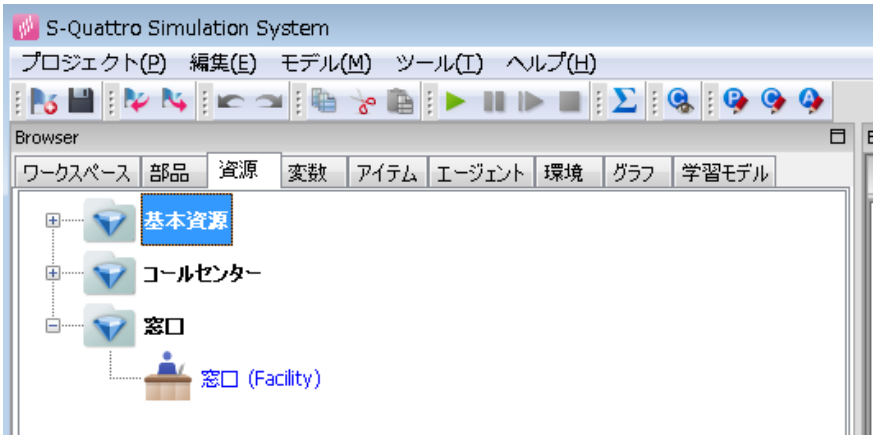


図 147: 窓口モデル用部品（資源タブ）

13.1　駅の窓口

テンプレートモデルの「駅の窓口」を見ていきましょう。このモデルは、次のような状況を想定しています。

- お客は駅の窓口にランダムに入店する

図 148: 窓口モデル用部品（部品タブ）

- 入店したお客は窓口で切符を購入する
- 全ての窓口が対応中の場合、お客は 1 列に並び窓口が空くのを待つ
- 窓口が空いたら、行列の先頭から順に窓口で切符を購入する
- 購入を終えたお客はそのまま退店する

シミュレーションを実行すると、行列の待ち人数、平均待ち時間、窓口の稼働時間、稼働率などの結果が出力されます。

13.2　テンプレートモデルのインポート

プロジェクトメニューから「プロジェクトをインポート」を選択し、(S-Quattro Simulation System のインストールフォルダ)¥samples¥駅の窓口(テンプレート).s4 を選択します。(図 149)

13.3　「駅の窓口」モデルを開く

ブラウザパネルのワークスペースタブにプロジェクトが表示されますので、モデルをダブルクリックします (図 150)。

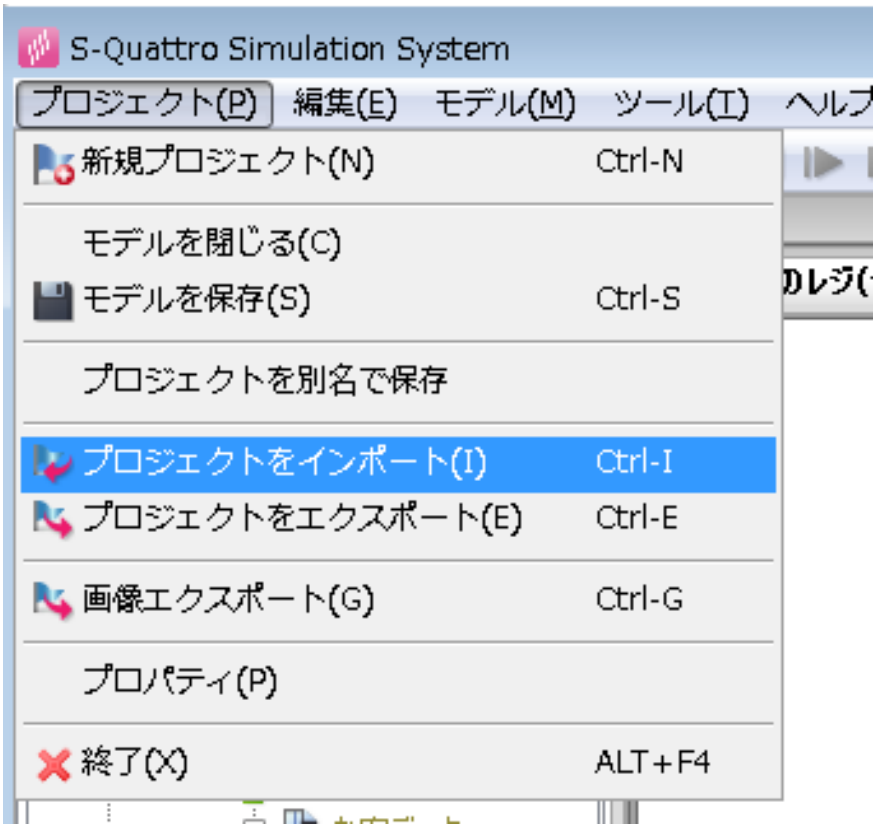


図 149: プロジェクトのインポート

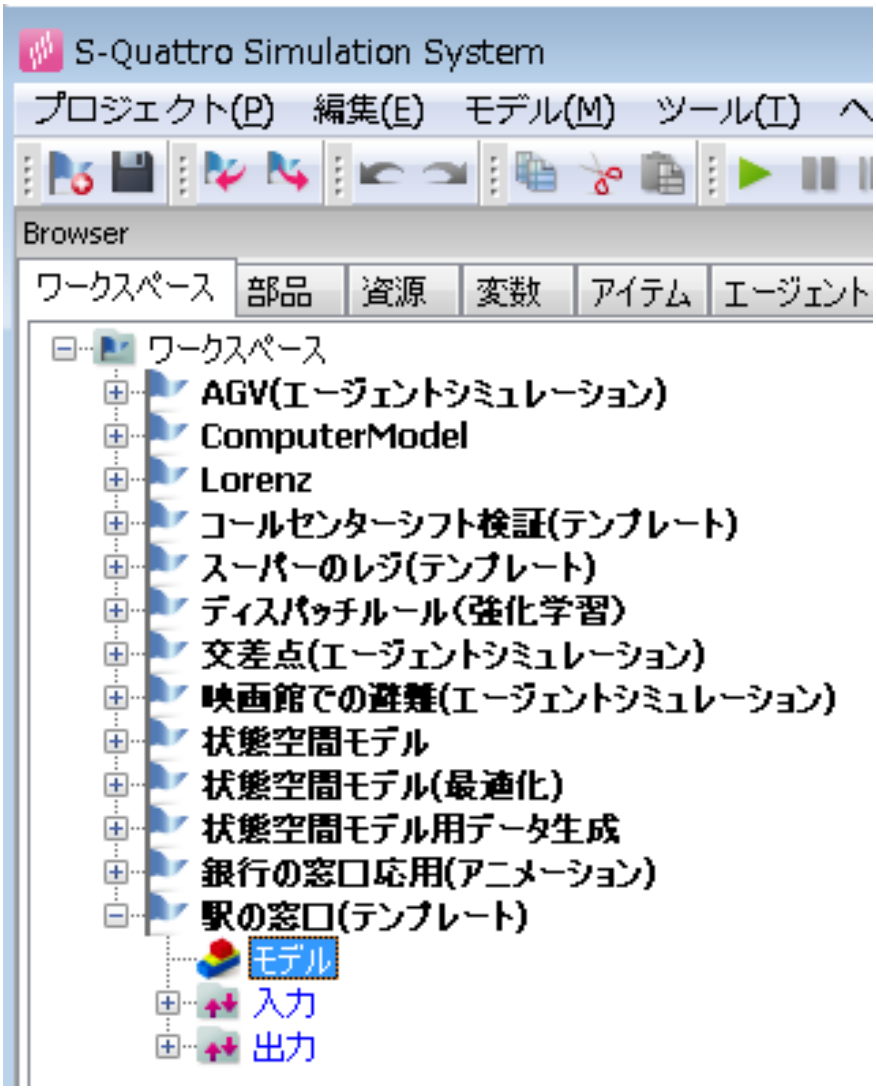

S-Quattro Simulation System
プロジェクト(P)
編集(E)
モデル(M)
ツール(T)
Browser
ワークスペース
部品
資源
変数
アイテム
エージェント
ワークスペース
AGV(エージェントシミュレーション)
ComputerModel
Lorenz
コールセンターシフト検証(テンプレート)
スーパーのレジ(テンプレート)
ディスパッチルール(強化学習)
交差点(エージェントシミュレーション)
映画館での避難(エージェントシミュレーション)
状態空間モデル
状態空間モデル(最適化)
状態空間モデル用データ生成
銀行の窓口応用(アニメーション)
駅の窓口(テンプレート)
モデル
入力
出力

図 150: モデルを開く

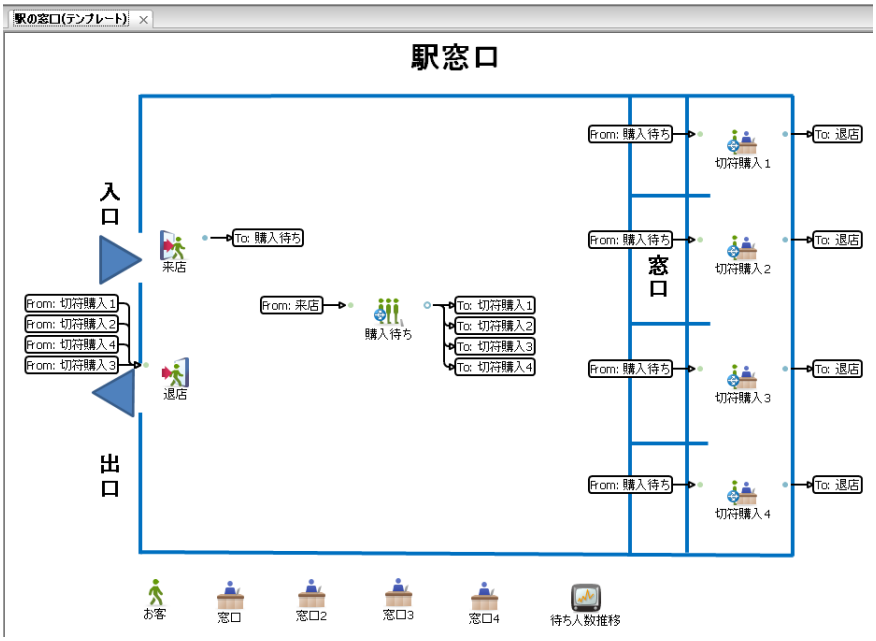


図 151: 駅の窓口モデル

13.4　「お客」部品

「お客」部品をダブルクリックすると、編集画面が開きますがここでは特に設定項目はありません。「お客」部品はアイテムタブの部品をそのまま用いています。

13.5　「来店」部品

「来店」部品では、お客が窓口に到着する時間間隔を設定しています。これは部品タブの「窓口到着」部品を用いていますが、部品の名前を「来店」に変えているのみです。「来店」部品をダブルクリックすると、部品編集画面が表示されます (図 152)。

来店開始時間には、常に 0 を指定しておきます。来店間隔には、お客が窓口に到着する間隔を設定します。ここでは、常に一定の間隔で到着する「固定」や、「指数分布」、「正規分布」、「アーラン分布」の確率分布に従うように設定が出来ます。テンプレートモデルでは時刻毎に指数分布のパラメータ (平均到着間隔) を変化させることが出来る「指数分布（パラメータ時間変化)」が設定されています。この分布のパラメータは入力フォルダへ表形式で与えておく必要があります。入力フォルダにデータを設定する方法については、操作マニュアルの「4.4 データのインポート」を参照してください。

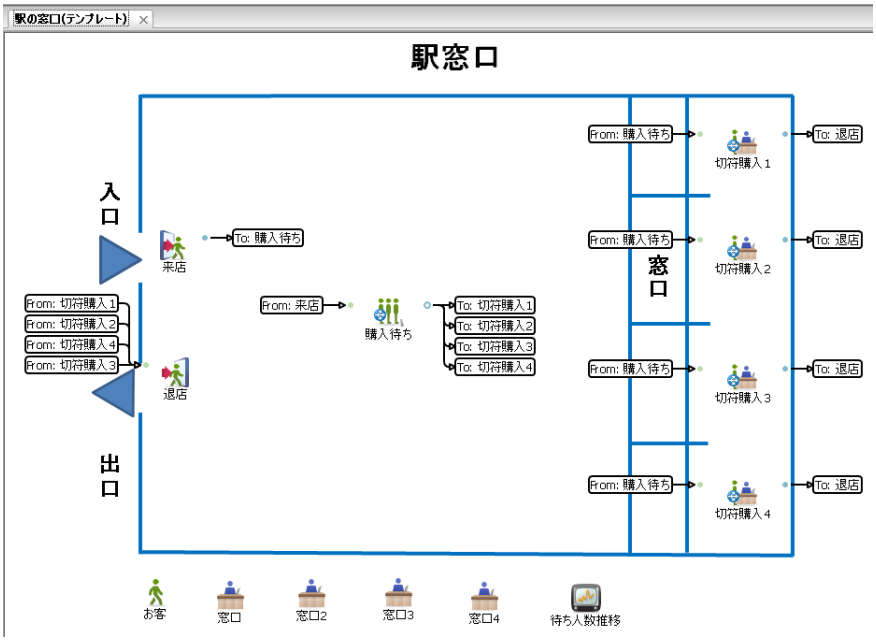


図 152: 来店部品

時間列には time 列、平均列には mean 列、単位には「秒」が設定されています。例えば図 150 の設定では、10:30:00〜11:59:59 の間、平均 3 分間隔の指数分布に従い、12:00:00〜13:09:59 の間、平均 1 分間隔の指数分布に従います。時間列には hh:mm:ss の形式で文字列として与えてください。2015-2-15 10:30:00 のように日付指定する事も可能です。この場合のフォーマットは yyyy-m-dd hh:mm:ss となります。日付を跨ぐシミュレーションの場合にはこのように日付も指定する必要があります。

13.6　「購入待ち」部品

「購入待ち」部品は、部品タブの「窓口選択部品：切符購入窓口」部品の名前を変えて用いています。ダブルクリックすると編集画面が開きますが、設定項目はありません。テンプレートモデルを編集される場合には、「窓口選択部品：切符購入窓口」のリンク先には必ず部品タブの「窓口利用部品：切符購入窓口」である必要があります。

13.7　「窓口」部品

「窓口」部品は、資源タブの「窓口」部品をそのまま用いています。この部品では窓口の開閉スケジュールを予め設定する事ができます。

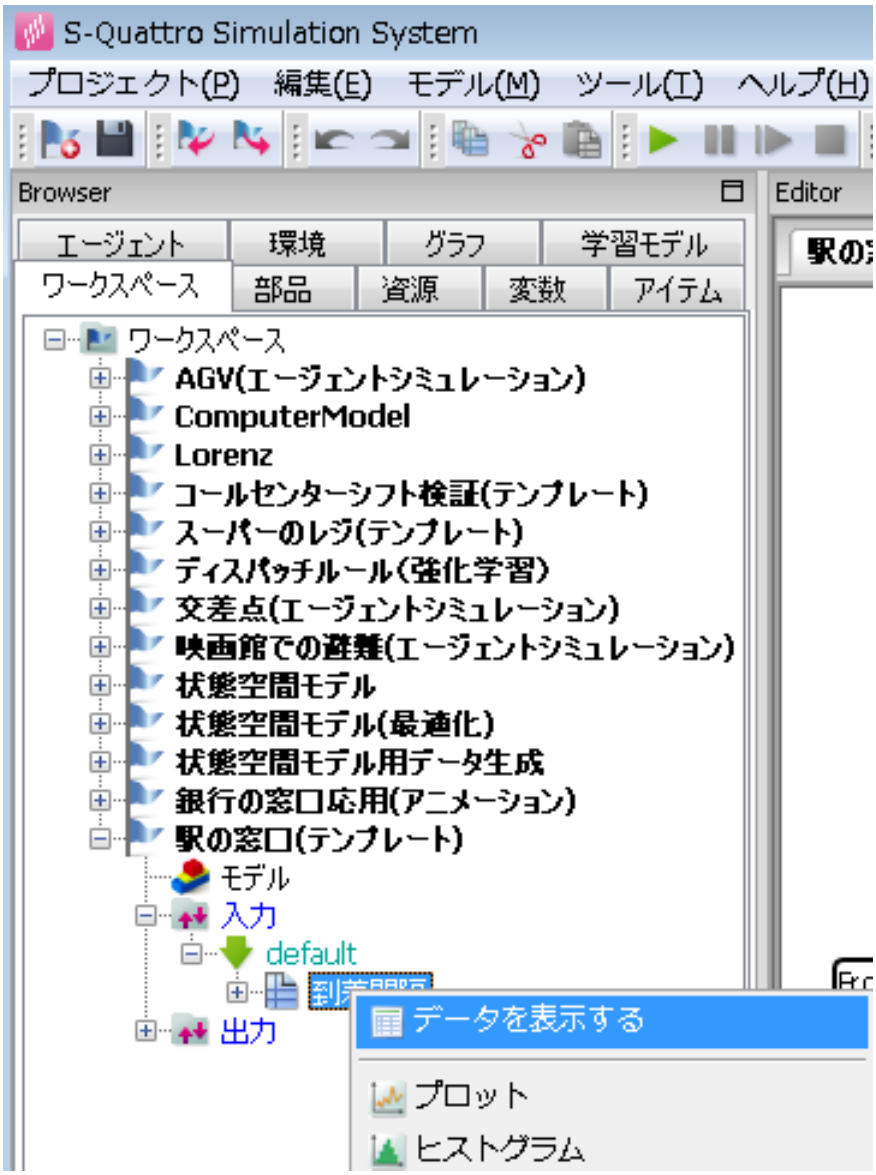


図 153: 入力データの確認


データ (到着...
time
mean
1
10:30:00
3.000
2
12:00:00
1.000
3
13:10:00
2.000
4
14:24:00
3.000
5
18:30:00
4.000
部分一致検索
テーブル：5 行 2 列

図 154: 平均到着間隔

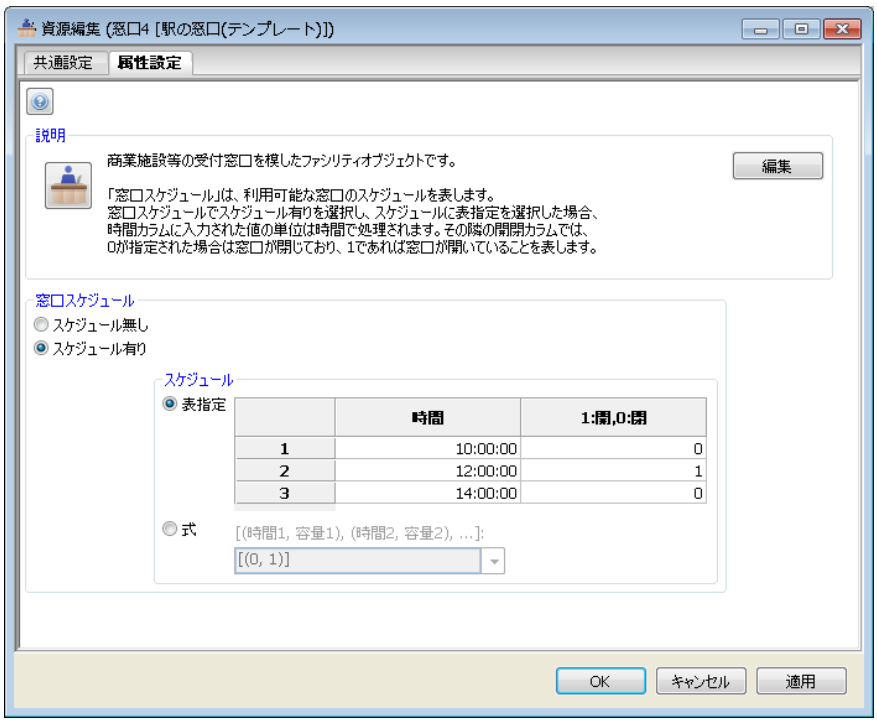


図 155: 窓口部品

図 155 の設定では、10:00:00 から 11:59:59 までは窓口が閉められ、12:00:00 から 13:59:59 までは窓口を開け、14:00:00 以降は再び窓口が閉められるスケジュールが設定されています。尚、日付を跨ぐシミュレーションの場合には日付も指定する必要があります。日付を指定する場合には、フォーマットは yyyy-m-dd hh:mm:ss となります。

13.8　「切符購入」部品

「切符購入」部品は、部品タブの「窓口利用部品：切符購入窓口」部品の名前を変えて用いています。この部品では、対応する窓口の設定とサービス時間を設定する事が出来ます。図 156 では、平均 5 分の指数分布に従う設定になっています。

13.9　「退店」部品

「退店」部品は部品タブの「対応終了」部品の名前を変えてそのまま用いています。ダブルクリックすると、編集画面が開きますが特に設定項目はありません。

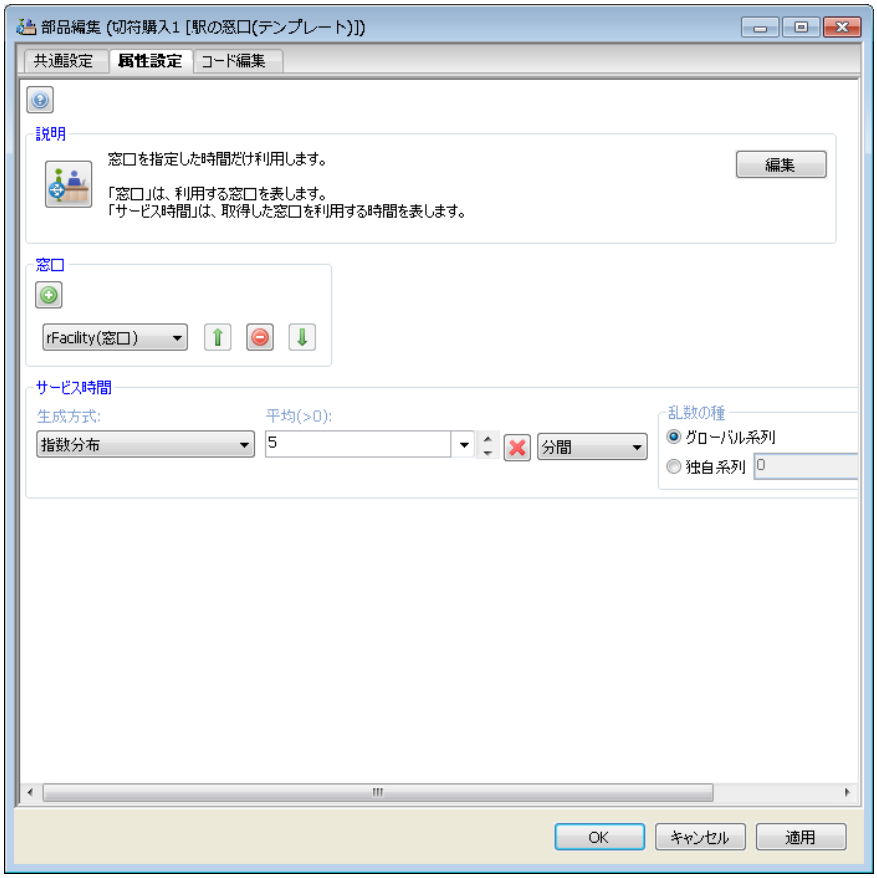

部品編集 (切符購入1 [駅の窓口(テンプレート)])
共通設定
属性設定
コード編集
説明
窓口を指定した時間だけ利用します。
「窓口」は、利用する窓口を表します。
「サービス時間」は、取得した窓口を利用する時間を表します。
編集
窓口
rFacility(窓口)
サービス時間
生成方式:
平均(>0):
指数分布
5
分間
乱数の種
グローバル系列
独自系列
0
OK
キャンセル
適用

図 156: 切符購入部品

13.10　シミュレーションパラメータ

シミュレーション開始時間とシミュレーション終了時間を設定します。図 157 の設定では、2016 年 2 月 16 日 10 時から 2016 年 2 月 16 日の 20 時までのシミュレーションをします。

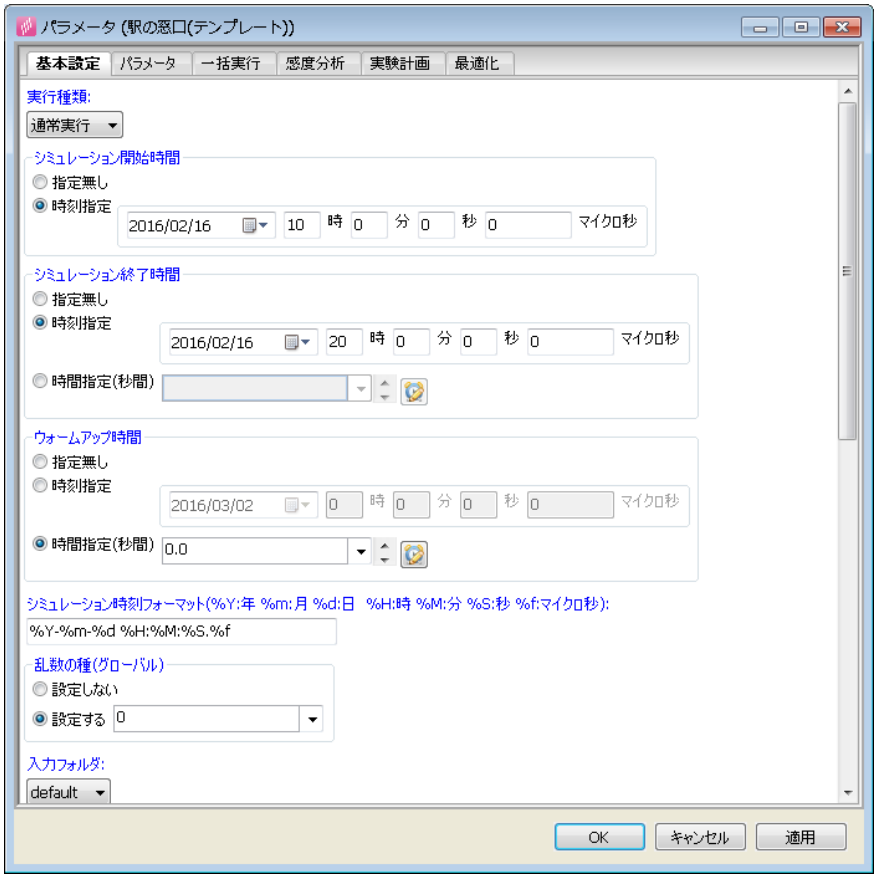


図 157: シミュレーションパラメータ

13.11　出力結果

シミュレーション実行後、出力フォルダには待ち人数の推移や各窓口の統計量などが出力されます（図 158）。「購入待ち-出力」には各時刻の「待ち人数」が記録されます（図 159）。「窓口」には、「利用フラグ」列にレジの利用状況 (1:利用中、0：空き) が出力されています。また、「窓口スケジュール」には、時刻毎の窓口の開閉スケジュールも出力されています。

各窓口の統計量には、各窓口の稼働時間、稼働率（稼働時間÷窓口が開いている時間）が出力されます。また、窓口全体の統計量には、全ての窓口に

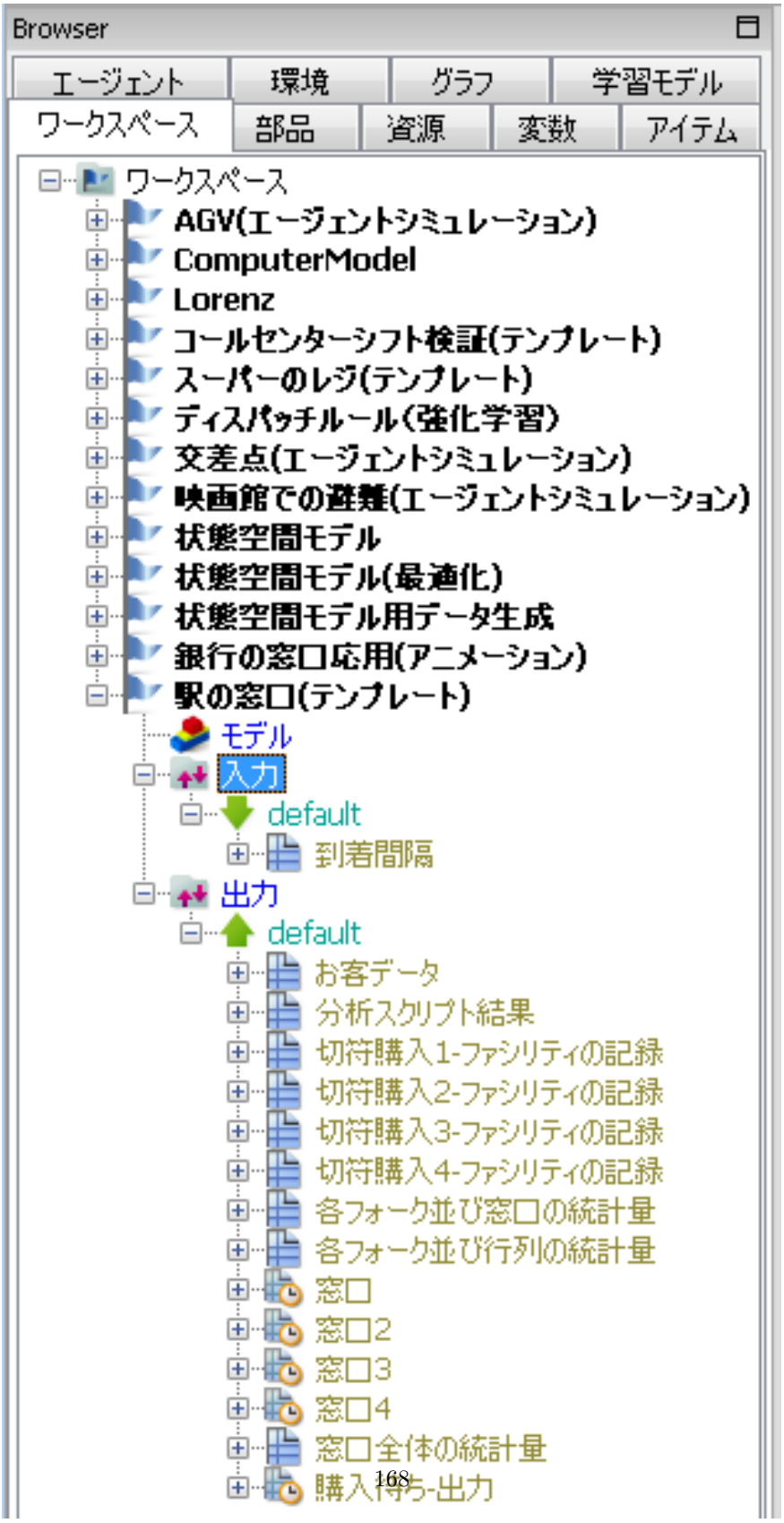
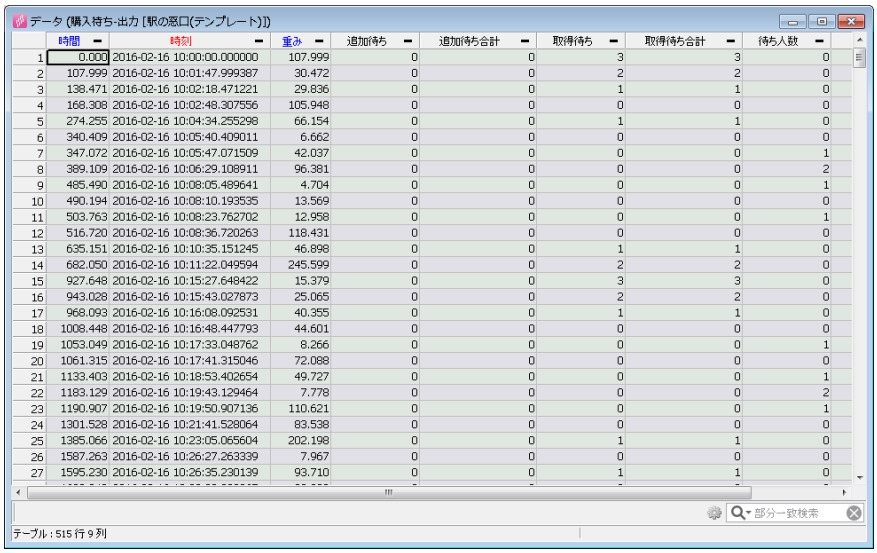


図 158: 駅の窓口の出力フォルダ

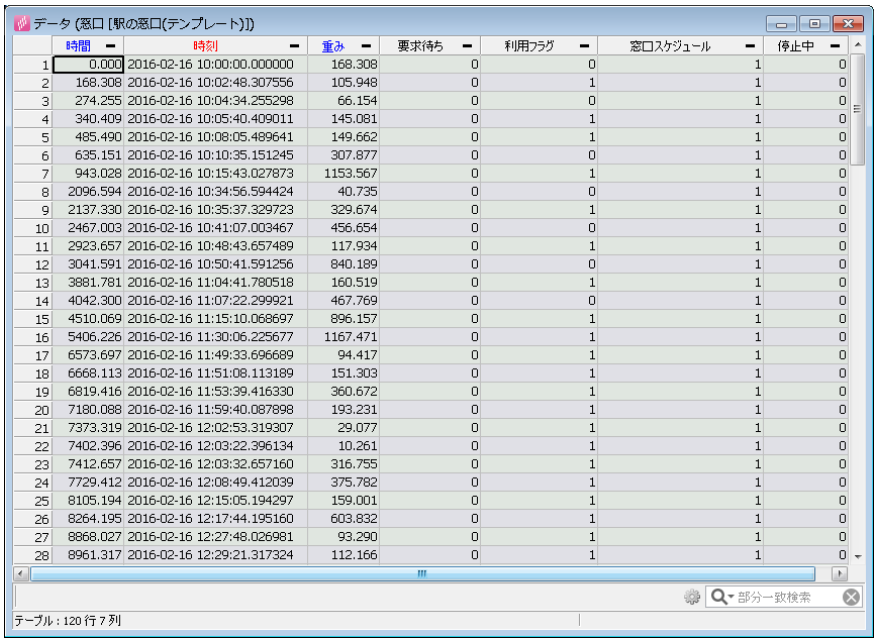

データ (購入待ち-出力 [駅の窓口(テンプレート)])

	時間	時刻	重み	追加待ち	追加待ち合計	取得待ち	取得待ち合計	待ち人数
1	0.000	2016-02-16 10:00:00.000000	107.999	0	0	3	3	0
2	107.999	2016-02-16 10:01:47.999387	30.472	0	0	2	2	0
3	138.471	2016-02-16 10:02:18.471221	29.836	0	0	1	1	0
4	168.308	2016-02-16 10:02:48.307556	105.948	0	0	0	0	0
5	274.255	2016-02-16 10:04:34.255298	66.154	0	0	1	1	0
6	340.409	2016-02-16 10:05:40.409011	6.662	0	0	0	0	0
7	347.072	2016-02-16 10:05:47.071509	42.037	0	0	0	0	1
8	389.109	2016-02-16 10:06:29.108911	96.381	0	0	0	0	2
9	485.490	2016-02-16 10:08:05.489641	4.704	0	0	0	0	1
10	490.194	2016-02-16 10:08:10.193535	13.569	0	0	0	0	0
11	503.763	2016-02-16 10:08:23.762702	12.958	0	0	0	0	1
12	516.720	2016-02-16 10:08:36.720263	118.431	0	0	0	0	0
13	635.151	2016-02-16 10:10:35.151245	46.898	0	0	1	1	0
14	682.050	2016-02-16 10:11:22.049594	245.599	0	0	2	2	0
15	927.648	2016-02-16 10:15:27.648422	15.379	0	0	3	3	0
16	943.028	2016-02-16 10:15:43.027873	25.065	0	0	2	2	0
17	968.093	2016-02-16 10:16:08.092531	40.355	0	0	1	1	0
18	1008.448	2016-02-16 10:16:48.447793	44.601	0	0	0	0	0
19	1053.049	2016-02-16 10:17:33.048762	8.266	0	0	0	0	1
20	1061.315	2016-02-16 10:17:41.315046	72.088	0	0	0	0	0
21	1133.403	2016-02-16 10:18:53.402654	49.727	0	0	0	0	1
22	1183.129	2016-02-16 10:19:43.129464	7.778	0	0	0	0	2
23	1190.907	2016-02-16 10:19:50.907136	110.621	0	0	0	0	1
24	1301.528	2016-02-16 10:21:41.528064	83.538	0	0	0	0	0
25	1385.066	2016-02-16 10:23:05.065604	202.198	0	0	1	1	0
26	1587.263	2016-02-16 10:26:27.263339	7.967	0	0	0	0	0
27	1595.230	2016-02-16 10:26:35.230139	93.710	0	0	1	1	0

部分一致検索

テーブル：515 行 9 列

図 159: 購入待ち結果


データ (窓口 [駅の窓口(テンプレート)])

	時間	時刻	重み	要求待ち	利用フラグ	窓口スケジュール	停止中
1	0.000	2016-02-16 10:00:00.000000	168.308	0	0	1	0
2	168.308	2016-02-16 10:02:48.307556	105.948	0	1	1	0
3	274.255	2016-02-16 10:04:34.255298	66.154	0	0	1	0
4	340.409	2016-02-16 10:05:40.409011	145.081	0	1	1	0
5	485.490	2016-02-16 10:08:05.489641	149.662	0	1	1	0
6	635.151	2016-02-16 10:10:35.151245	307.877	0	0	1	0
7	943.028	2016-02-16 10:15:43.027873	1153.567	0	1	1	0
8	2096.594	2016-02-16 10:34:56.594424	40.735	0	0	1	0
9	2137.330	2016-02-16 10:35:37.329723	329.674	0	1	1	0
10	2467.003	2016-02-16 10:41:07.003467	456.654	0	0	1	0
11	2923.657	2016-02-16 10:48:43.657489	117.934	0	1	1	0
12	3041.591	2016-02-16 10:50:41.591256	840.189	0	0	1	0
13	3881.781	2016-02-16 11:04:41.780518	160.519	0	1	1	0
14	4042.300	2016-02-16 11:07:22.299921	467.769	0	0	1	0
15	4510.069	2016-02-16 11:15:10.068697	896.157	0	1	1	0
16	5406.226	2016-02-16 11:30:06.225677	1167.471	0	1	1	0
17	6573.697	2016-02-16 11:49:33.696689	94.417	0	1	1	0
18	6668.113	2016-02-16 11:51:08.113189	151.303	0	1	1	0
19	6819.416	2016-02-16 11:53:39.416330	360.672	0	1	1	0
20	7180.088	2016-02-16 11:59:40.087898	193.231	0	1	1	0
21	7373.319	2016-02-16 12:02:53.319307	29.077	0	1	1	0
22	7402.396	2016-02-16 12:03:22.396134	10.261	0	1	1	0
23	7412.657	2016-02-16 12:03:32.657160	316.755	0	1	1	0
24	7729.412	2016-02-16 12:08:49.412039	375.782	0	1	1	0
25	8105.194	2016-02-16 12:15:05.194297	159.001	0	1	1	0
26	8264.195	2016-02-16 12:17:44.195160	603.832	0	1	1	0
27	8868.027	2016-02-16 12:27:48.026981	93.290	0	1	1	0
28	8961.317	2016-02-16 12:29:21.317324	112.166	0	1	1	0

部分一致検索

テーブル：120 行 7 列

図 160: 窓口結果

対する統計量が計算されています。平均滞留時間は、お客が来店してから退店するまでの時間の平均値です。

図 161: 各窓口の統計量

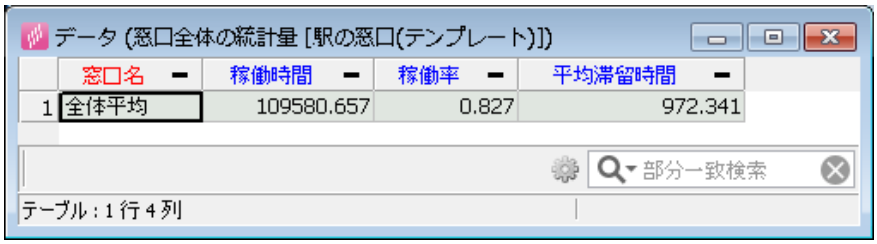


図 162: 窓口全体の統計量

13.12　スーパーのレジ

窓口モデル用の部品を利用すると、スーパーマーケットで買い物をしたお客がレジに並ぶ様子をシミュレーションするようなモデルも作る事が出来ます。テンプレートモデルの「スーパーのレジ」を見ていきましょう。

- 商品を選んだお客はランダムにレジに到着する
- レジに到着したお客は最も待ち人数が少ないレジを選択して各レジに並ぶ
- レジは予め決められたスケジュールに応じて開閉される
- 閉じているレジにはお客は並べない
- レジの所要時間はランダムである

- レジで会計を終えたお客はそのまま退店する

レジは複数台ありますので、レジ毎に行列が出来ます。テンプレートモデルでは、各レジの待ち人数、平均待ち時間、稼働時間、稼働率などがシミュレーションできます。

13.13 テンプレートプロジェクトのインポート

プロジェクトメニューから「プロジェクトをインポート」を選択し、(S-Quattro Simulation System のインストールフォルダ)¥samples¥スーパーのレジ (テンプレート).s4 を選択します (図 163)。

図 163: プロジェクトのインポート

13.14 スーパーのレジモデルを開く

ブラウザパネルのワークスペースタブに作成されたプロジェクトが表示されますので、モデルをダブルクリックします ()。

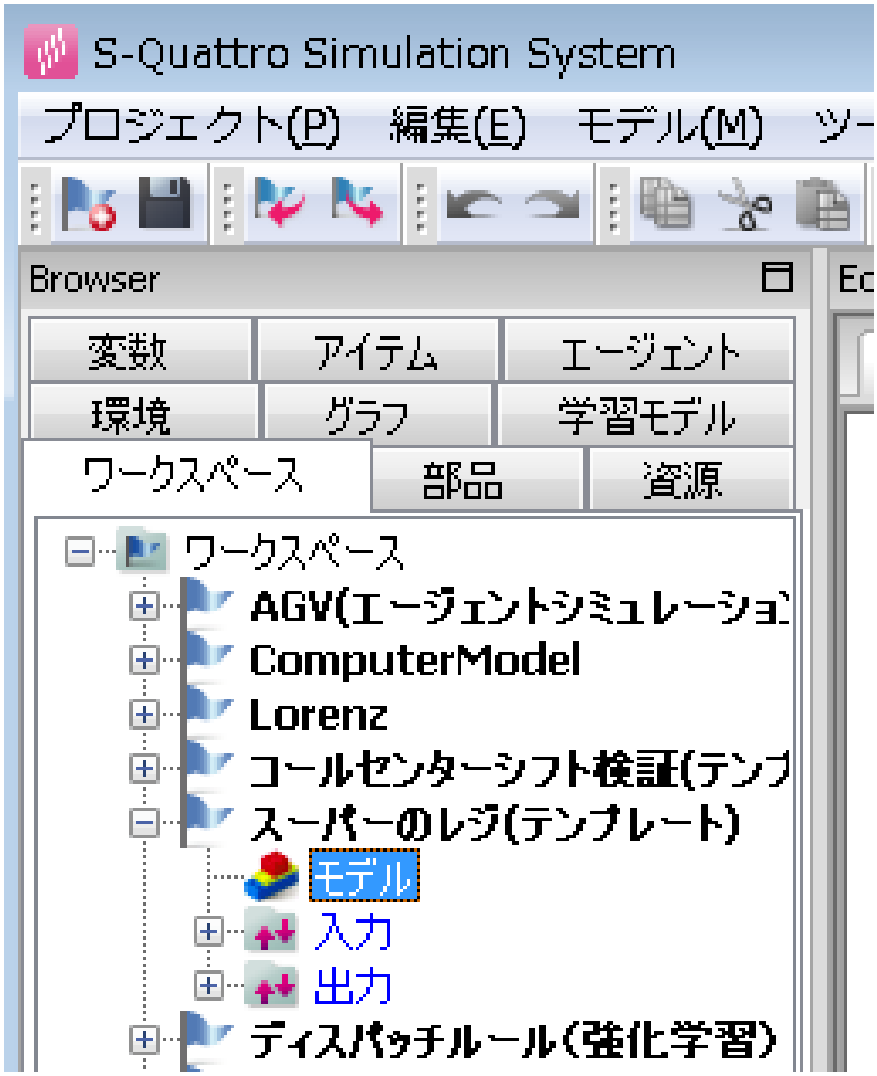


図 164: プロジェクトのインポート

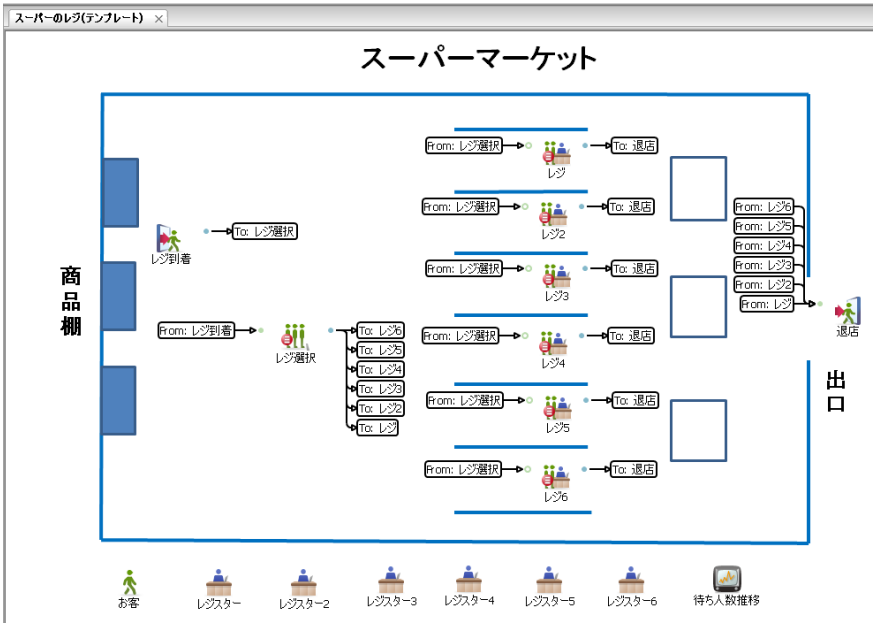


図 165: スーパーのレジモデル

13.15　「お客」部品

「お客」部品はアイテムタブの部品をそのまま用いています。

13.16　「レジ到着」部品

「レジ到着」部品では、お客がレジに到着する時間間隔を設定しています。これは部品タブの「窓口到着」部品を用い、名前「レジ到着」に変更しています。「レジ到着」部品をダブルクリックすると、部品編集画面が表示されます。 設定は「駅の窓口」モデルと同様です。

入力データの確認方法や、平均到着間隔データのフォーマットは「駅の窓口」と同様です。

13.17　「レジ選択」部品

レジ選択」部品は「窓口選択部品：レジ」を用いています。共通設定タブの出力ポートでお客のレジの選択方法が設定されています。「選択方式」に「最短キュー」を設定するとお客は最も行列の短いレジを選択して並びます。尚、窓口選択部品のリンク先には必ず「窓口利用部品：レジ」である必要があります。テンプレートモデルを編集される場合には、注意してください。

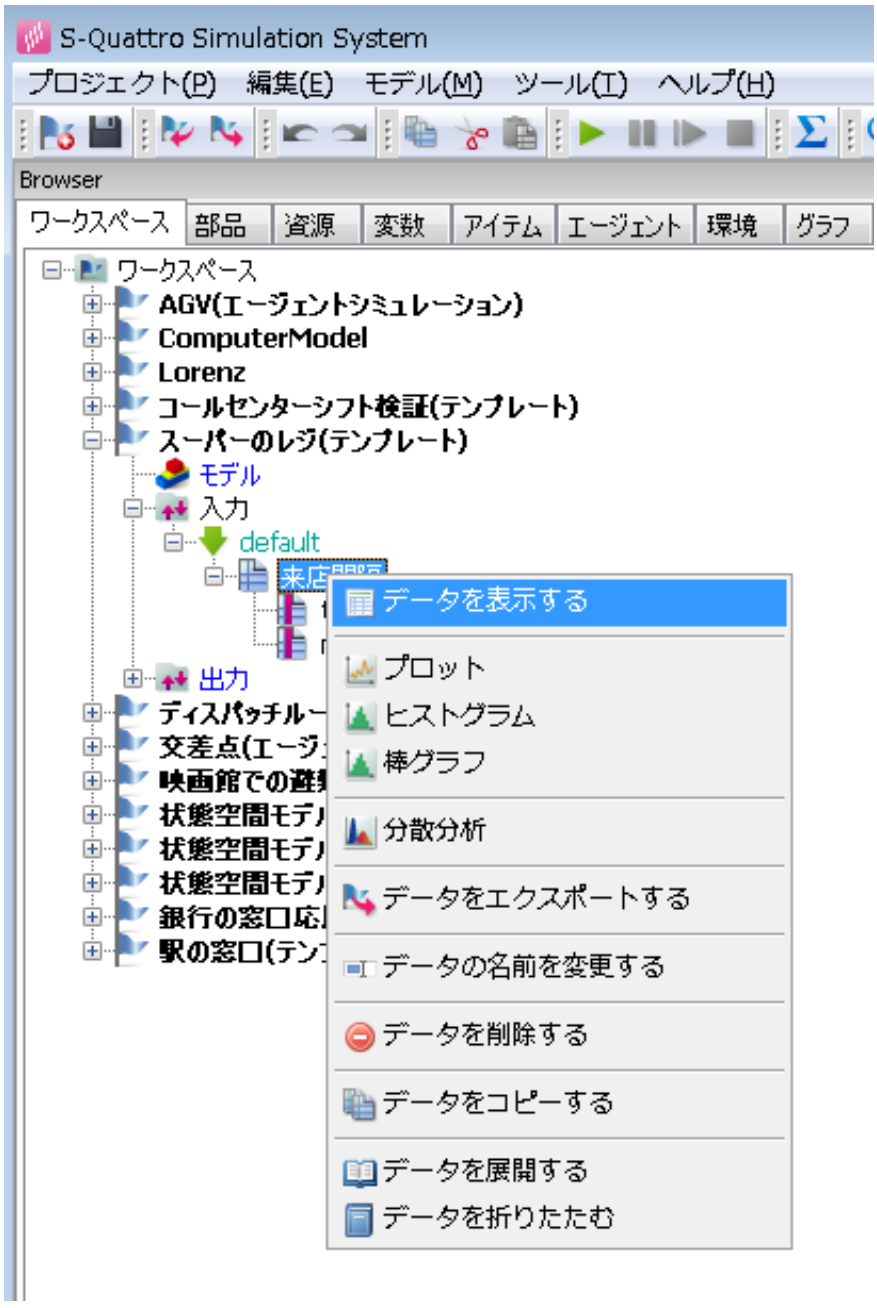


図 166: 入力データの確認


データ (来店間隔 [ス...
time
mean
1 10:00:00 60.000
2 11:00:00 45.000
3 12:00:00 30.000
4 14:00:00 40.000
5 15:00:00 60.000
6 18:00:00 50.000
7 20:00:00 150.000
部分一致検索
テーブル：7 行 2 列

図 167: 平均到着間隔

13.18　「レジスター」部品

「レジスター」部品は資源タブの「窓口」部品の名前を変えて用いています。レジの開閉スケジュールを予め設定します。設定方法は「駅の窓口」と同様です。

図 168: 窓口部品

13.19　「レジ」部品

「レジ」部品は部品タブの「窓口利用部品：レジ」部品の名前を変えて用いています。対応するレジスターの設定と、所要時間を設定する事が出来ます。設定方法は「駅の窓口」と同様です。

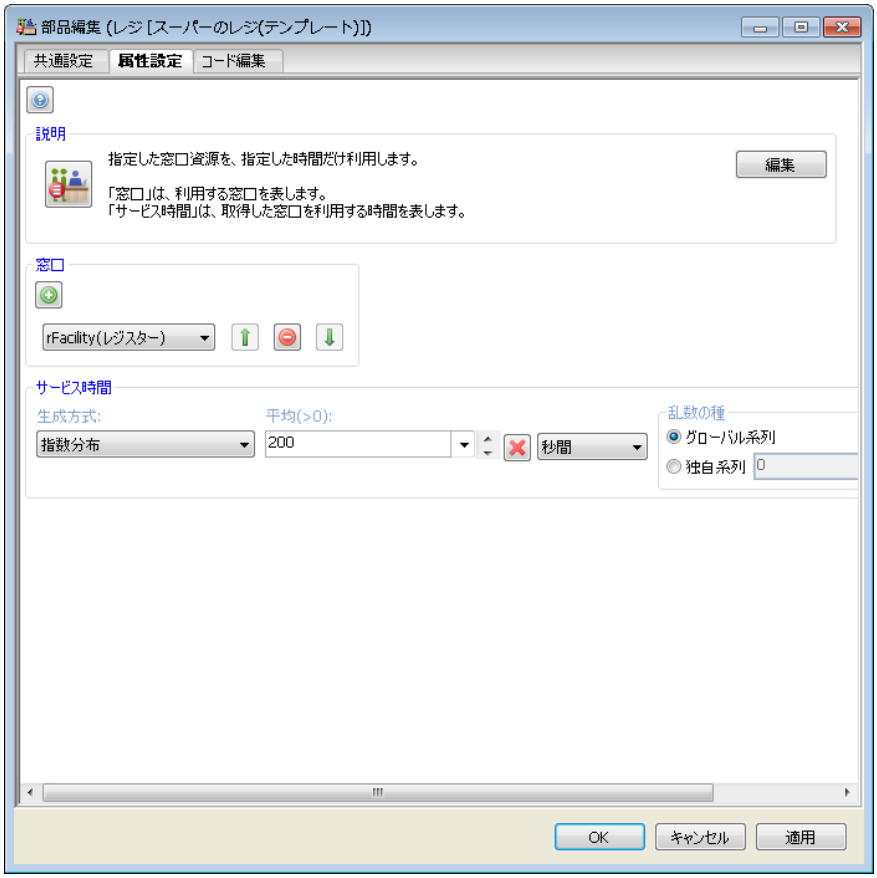

部品編集 (レジ [スーパーのレジ(テンプレート)])
共通設定
属性設定
コード編集
説明
指定した窓口資源を、指定した時間だけ利用します。
「窓口」は、利用する窓口を表します。
「サービス時間」は、取得した窓口を利用する時間を表します。
編集
窓口
rFacility(レジスター)
サービス時間
生成方式:
平均(>0):
指数分布
200
秒間
乱数の種
グローバル系列
独自系列
0
OK
キャンセル
適用

図 169: レジ部品

13.20　「退店」部品

「退店」部品は部品タブの「対応終了」部品の名前を変えて用いています。ダブルクリックすると、編集画面が開きますが特に設定項目はありません。

13.21　出力結果

「駅の窓口」と同様、シミュレーション実行後、出力フォルダには各レジの待ち人数の推移や各レジの統計量などが出力されます。

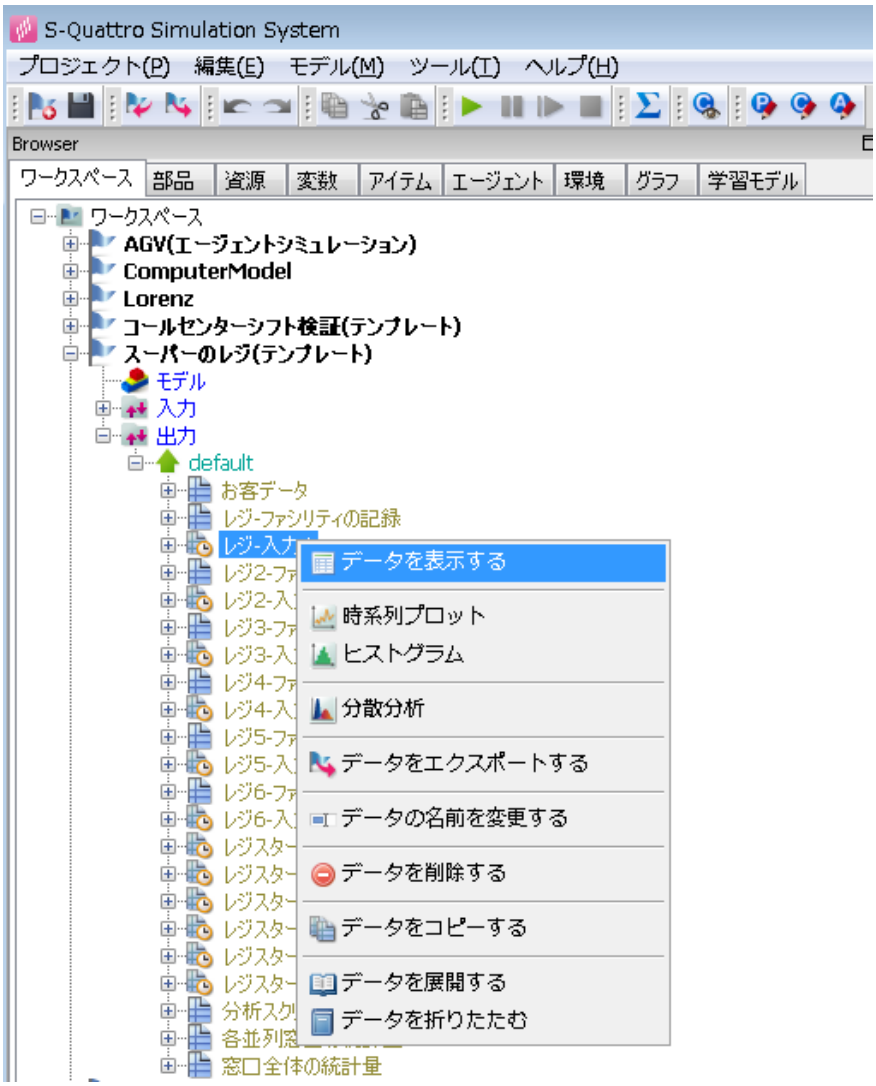
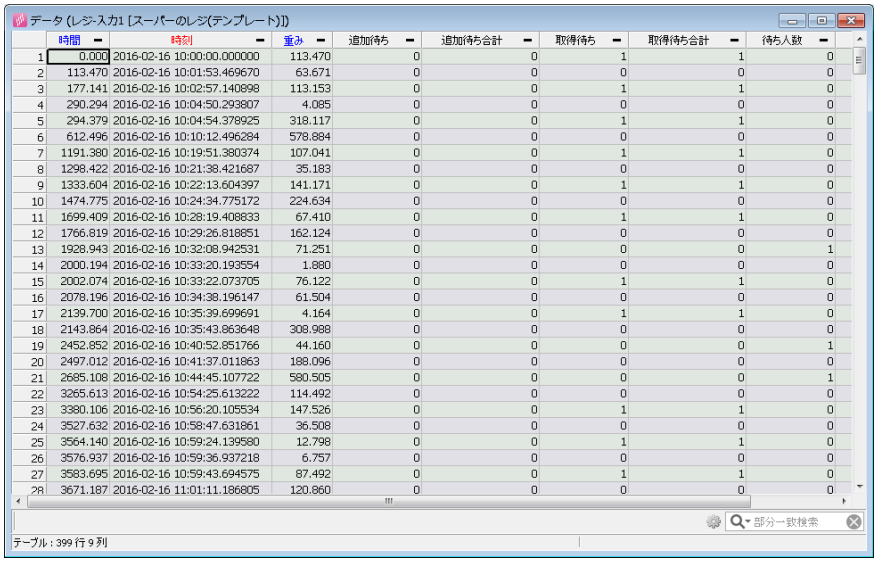


図 170: スーパーのレジ出力フォルダ

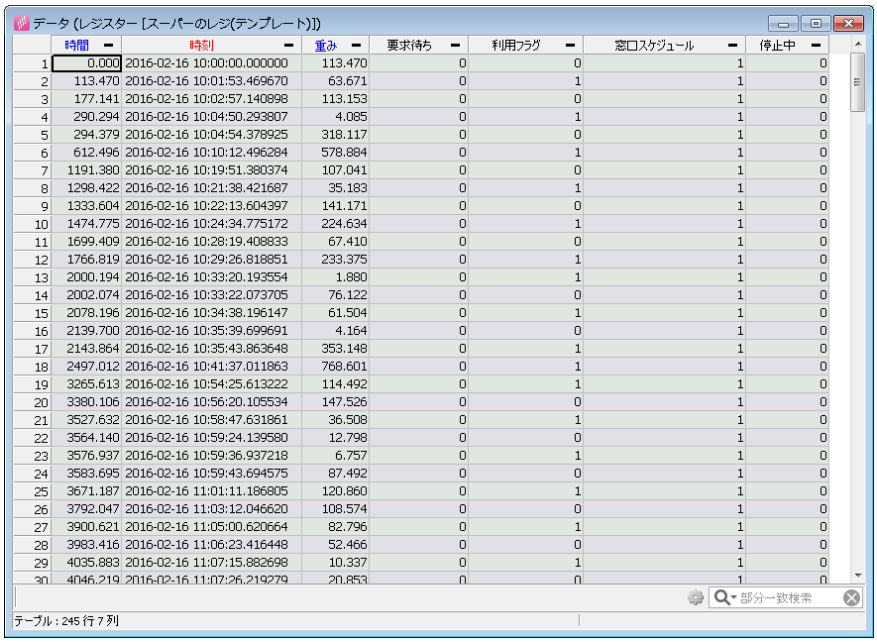

データ (レジ-スカ1 [スーパーのレジ(テンプレート)])

	時間	時刻	重み	追加待ち	追加待ち合計	取得待ち	取得待ち合計	待ち人数
1	0.000	2016-02-16 10:00:00.000000	113.470	0	0	1	1	0
2	113.470	2016-02-16 10:01:53.469670	63.671	0	0	0	0	0
3	177.141	2016-02-16 10:02:57.140898	113.153	0	0	1	1	0
4	290.294	2016-02-16 10:04:50.293807	4.085	0	0	0	0	0
5	294.379	2016-02-16 10:04:54.378925	318.117	0	0	1	1	0
6	612.496	2016-02-16 10:10:12.496284	578.884	0	0	0	0	0
7	1191.380	2016-02-16 10:19:51.380374	107.041	0	0	1	1	0
8	1298.422	2016-02-16 10:21:38.421687	35.183	0	0	0	0	0
9	1333.604	2016-02-16 10:22:13.604397	141.171	0	0	1	1	0
10	1474.775	2016-02-16 10:24:34.775172	224.634	0	0	0	0	0
11	1699.409	2016-02-16 10:28:19.408833	67.410	0	0	1	1	0
12	1766.819	2016-02-16 10:29:26.818851	162.124	0	0	0	0	0
13	1928.943	2016-02-16 10:32:08.942531	71.251	0	0	0	0	1
14	2000.194	2016-02-16 10:33:20.193554	1.880	0	0	0	0	0
15	2002.074	2016-02-16 10:33:22.073705	76.122	0	0	1	1	0
16	2078.196	2016-02-16 10:34:38.196147	61.504	0	0	0	0	0
17	2139.700	2016-02-16 10:35:39.699691	4.164	0	0	1	1	0
18	2143.864	2016-02-16 10:35:43.863648	308.988	0	0	0	0	0
19	2452.852	2016-02-16 10:40:52.851766	44.160	0	0	0	0	1
20	2497.012	2016-02-16 10:41:37.011863	188.096	0	0	0	0	0
21	2685.108	2016-02-16 10:44:45.107722	580.505	0	0	0	0	1
22	3265.613	2016-02-16 10:54:25.613222	114.492	0	0	0	0	0
23	3380.106	2016-02-16 10:56:20.105534	147.526	0	0	1	1	0
24	3527.632	2016-02-16 10:58:47.631861	36.508	0	0	0	0	0
25	3564.140	2016-02-16 10:59:24.139580	12.798	0	0	1	1	0
26	3576.937	2016-02-16 10:59:36.937218	6.757	0	0	0	0	0
27	3583.695	2016-02-16 10:59:43.694575	87.492	0	0	1	1	0
28	3671.187	2016-02-16 11:01:11.186805	120.860	0	0	0	0	0

部分一致検索

テーブル：399 行 9 列

図 171: レジ結果


データ (レジスター [スーパーのレジ(テンプレート)])

	時間	時刻	重み	要求待ち	利用フラグ	窓口スケジュール	停止中
1	0.000	2016-02-16 10:00:00.000000	113.470	0	0	1	0
2	113.470	2016-02-16 10:01:53.469670	63.671	0	1	1	0
3	177.141	2016-02-16 10:02:57.140898	113.153	0	0	1	0
4	290.294	2016-02-16 10:04:50.293807	4.085	0	1	1	0
5	294.379	2016-02-16 10:04:54.378925	318.117	0	0	1	0
6	612.496	2016-02-16 10:10:12.496284	578.884	0	1	1	0
7	1191.380	2016-02-16 10:19:51.380374	107.041	0	0	1	0
8	1298.422	2016-02-16 10:21:38.421687	35.183	0	1	1	0
9	1333.604	2016-02-16 10:22:13.604397	141.171	0	0	1	0
10	1474.775	2016-02-16 10:24:34.775172	224.634	0	1	1	0
11	1699.409	2016-02-16 10:28:19.408833	67.410	0	0	1	0
12	1766.819	2016-02-16 10:29:26.818851	233.375	0	1	1	0
13	2000.194	2016-02-16 10:33:20.193554	1.880	0	1	1	0
14	2002.074	2016-02-16 10:33:22.073705	76.122	0	0	1	0
15	2078.196	2016-02-16 10:34:38.196147	61.504	0	1	1	0
16	2139.700	2016-02-16 10:35:39.699691	4.164	0	0	1	0
17	2143.864	2016-02-16 10:35:43.863648	353.148	0	1	1	0
18	2497.012	2016-02-16 10:41:37.011863	768.601	0	1	1	0
19	3265.613	2016-02-16 10:54:25.613222	114.492	0	1	1	0
20	3380.106	2016-02-16 10:56:20.105534	147.526	0	0	1	0
21	3527.632	2016-02-16 10:58:47.631861	36.508	0	1	1	0
22	3564.140	2016-02-16 10:59:24.139580	12.798	0	0	1	0
23	3576.937	2016-02-16 10:59:36.937218	6.757	0	1	1	0
24	3583.695	2016-02-16 10:59:43.694575	87.492	0	0	1	0
25	3671.187	2016-02-16 11:01:11.186805	120.860	0	1	1	0
26	3792.047	2016-02-16 11:03:12.046620	108.574	0	0	1	0
27	3900.621	2016-02-16 11:05:00.620664	82.796	0	1	1	0
28	3983.416	2016-02-16 11:06:23.416448	52.466	0	0	1	0
29	4035.883	2016-02-16 11:07:15.882698	10.337	0	1	1	0
30	4046.219	2016-02-16 11:07:26.219279	20.853	0	0	1	0

部分一致検索

テーブル：245 行 7 列

図 172: レジスター結果

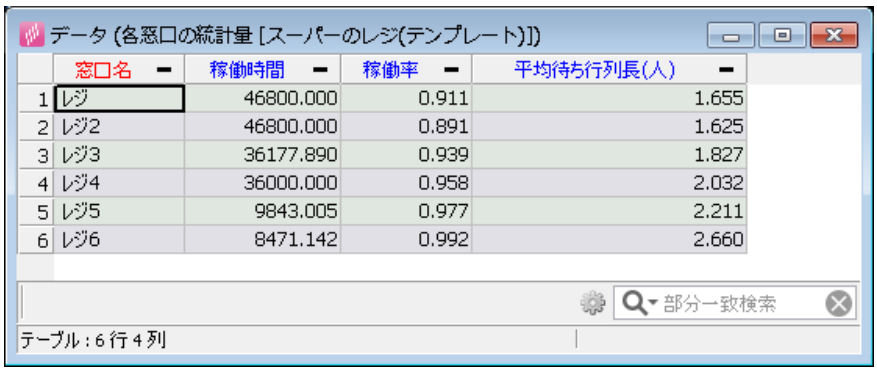


図 173: 各レジの統計量

図 174: レジ全体の統計量

以上、窓口モデル用の部品とそのテンプレートモデルの説明をしましたが、テンプレートモデルのモデルや、パラメータ値は編集する事ができます。また、窓口モデル用の部品を使って新しくシミュレーションモデルを作る事も出来ますので、お試しください。

14 OpenStreetMap を用いたネットワークシミュレーション

14.1 プロジェクトの準備

プロジェクトの新規作成 (Ctrl-N もしくは左上ボタン) より、任意のプロジェクト名を入力します。

ブラウザパネルの「環境」タブより「NW 地図」アイコンを、「エージェント」タブより「NW エージェント」アイコンをそれぞれプロジェクトにドラッグ・アンド・ドロップします。

「NW エージェント」編集画面の「環境オブジェクト」から「eNMMap(NW 地図)」を選択します。

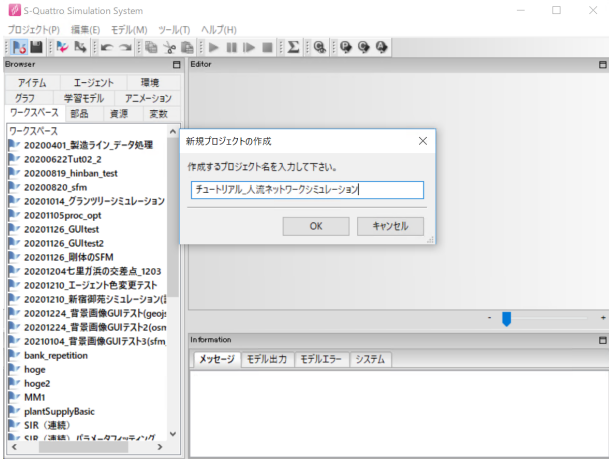


図 175: プロジェクトの作成

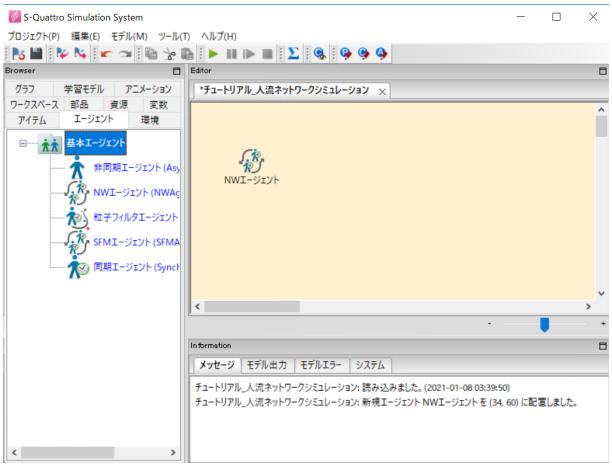


図 176: NWAgent の配置

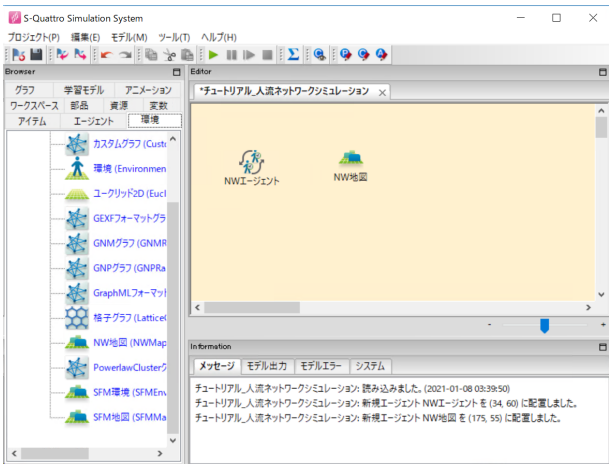


図 177: NWMap の配置

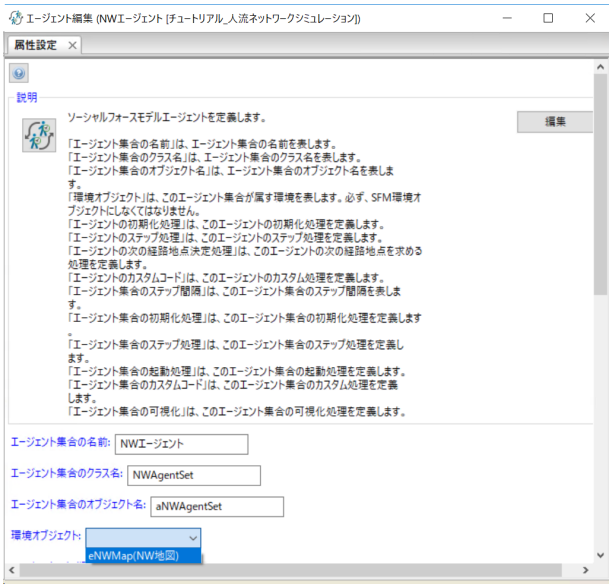


図 178: NWMap の選択

14.2 OpenStreetMap からネットワークデータを読み込む

本節では readOSM() メソッドを用いて現実の道路ネットワークを S4 上に読み込む方法を説明します。

14.2.1 データの準備

OpenStreetMap(以下 OSM) から道路ネットワーク情報をダウンロードします。OSM の地図 GUI https://www.openstreetmap.org/ からシミュレーションしたい範囲を選んでエクスポートし、得られた”map.osm”ファイルをプロジェクトフォルダ以下の input/default に配置します。

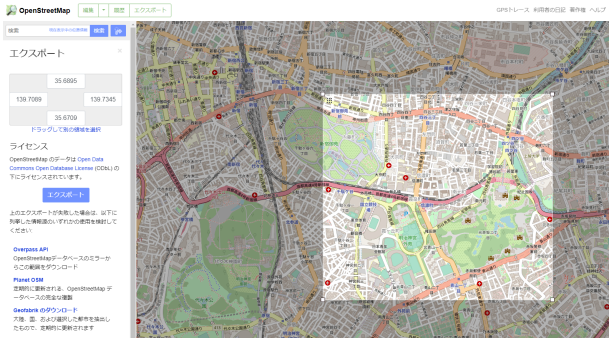


図 179: OSM 地図の取得

注意: 広範囲のシミュレーションを扱いたい場合、上記の方法ではエクスポートできない場合があります。その場合は http://download.geofabrik.de/ から該当地域 (Asia → Japan → kanto など) のページに進み、”kanto-latest.osm.pbf”をダウンロードし (関東地方の場合)、プロジェクトフォルダ以下の input/default に配置します。

図 180: 広範囲地図の取得

ヒント: プロジェクトフォルダはワークスペースタブのプロジェクト名を右クリックから「エクスプローラで表示」を選ぶことで開くことができます。

14.2.2 環境部品から読み込み

「NW 地図」部品の編集画面「環境の初期化後の処理」において、readOSM() メソッドを呼び出します。引数には (経度最小値, 緯度最小値, 経度最大値, 緯度最大値) を指定します。ここでは、OSM の HP で範囲指定した際に表示された値を入力します。

環境の初期化後の処理

```
self.readOSM(
  "map.osm",
  139.7089, 35.6709, 139.7345, 35.6895
)
```

注意: このネットワーク読み込み処理はシミュレーション実行時に行われるものであり、この内容を地図エディタで編集することはできません。

14.2.3 テスト実行

この時点で実行ボタンを押すとシミュレーションが実行され、読み込んだ道路ネットワーク上をランダムに生成されたエージェントが動き回る様子が確認できます。背景地図も表示されます。

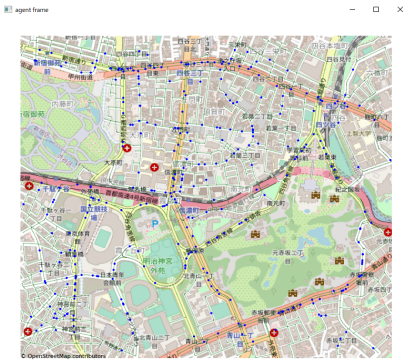


図 181: テスト実行

ヒント: 環境部品の編集画面より、背景地図の濃さ (不透明度) や人口カウントのヒートマップの可視化切り替えを行うことができます。

ヒント: 上の例ではエージェントは青色の点として描画されていますが、点の大きさを変えるには環境部品の編集画面より「エージェントパラメータのデフォルト値」→「表示半径」を変更します。

14.3 エージェント出現位置、移動方法の指定

上記のテスト実行ではエージェントの出現位置や移動先がランダムなものでした。本節ではエージェントの出現位置を指定したり、setDestination() メソッドや setTravel() メソッドなどのエージェント API を用いて移動先を指定したりする方法を説明していきます。

14.3.1 エージェントの出現位置を指定する

本節ではエージェントの出現位置を変更する方法を説明します。

例として、ここではエージェントの出現位置を信濃町駅近辺に変更します。

信濃町近辺の経路地点番号を知る方法はいくつかありますが、ここでは環境部品の nearestPathPoint() メソッドを使用する方法を説明します。

まず、「NW 地図部品」の「環境の初期化後の処理」において、nearestPathPoint() メソッドを呼び出します。引数には（信濃町駅の経度、信濃町駅の緯度)、キーワード引数 proj に”EPSG:2451”を指定します。proj を指定した場合、nearestPathPoint() メソッドの返り値は指定した経度、緯度に最も近い経路地点の経路地点番号 (つまり信濃町駅に最も近い経路地点の番号) となります。これを変数 shinanomachi としておきます。

環境の初期化後の処理

```
self.readOSM(
  "map.osm",
  139.7089, 35.6709, 139.7345, 35.6895
)

# (139.7202, 35.6800) は信濃町駅の経度緯度
self.shinanomachi = self.nearestPathPoint(
                      (139.7202, 35.6800),
                      proj="EPSG:2451"
                    )
```

信濃町駅近辺の経路地点番号を取得できたので、次にエージェントの出現位置を今設定した shinanomachi にしていきます。「NW エージェント」部品の「エージェント集合の初期化処理」においてデフォルトでは start=next(sample(vs)) となっている部分がエージェントの出現位置になりますが、ここを start=self.env.shinanomachi に変更します。

この時点で実行ボタンを押すとシミュレーションが実行され、エージェントが信濃町駅近辺から出現して動き回る様子が確認できます。

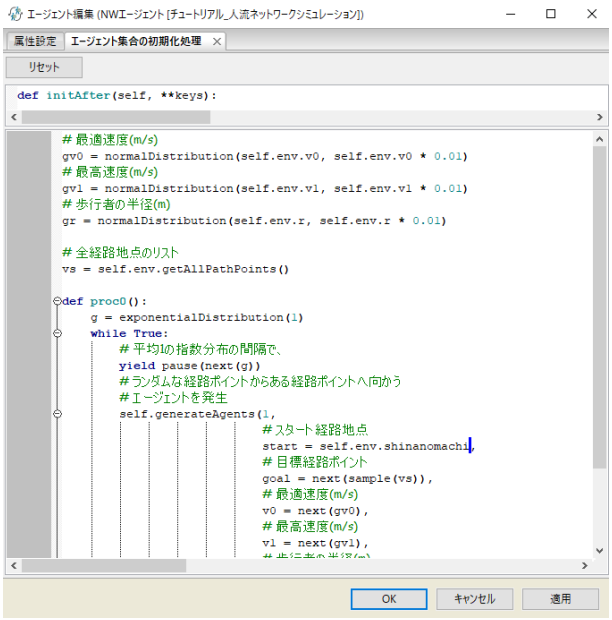


図 182: エージェントの出現位置の変更

14.3.2 エージェントの目的地を設定する。

前節でエージェントが信濃町駅から移動を開始するようにできましたが、移動先がランダムなままとなっています。そこで、次はエージェントの目的地を設定する方法を説明します。

例として目的地を四ツ谷駅近辺とします。前節では nearestPathPoint() メソッドを用いて信濃町駅に近い経路地点を 1 つ取得してエージェントの出現位置としましたが、ここでは innerPathPoints() メソッドを用いて四ツ谷駅近辺の経路地点を複数取得し、エージェントごとに目的経路地点をランダムに設定する方法を説明します。

「NW 地図部品」の「環境の初期化後の処理」において、innerPathPoints() メソッドを呼び出して指定した範囲 (今回は四ツ谷駅近辺) の経路地点のリストを取得し、変数 yotsuyaPoints としておきます。引数は (経度最小値, 緯度最小値, 経度最大値, 緯度最大値)、キーワード引数 proj に”EPSG:2451” を指定します。

環境の初期化後の処理

```
self.readOSM(
  "map.osm",
  139.7089, 35.6709, 139.7345, 35.6895
)

# (139.7202, 35.6800) は信濃町駅の経度、緯度
self.shinanomachi = self.nearestPathPoint(
                      (139.7202, 35.6800),
                      proj="EPSG:2451"
                    )
# (139.7268, 35.6828, 139.7326, 35.6879) は四ツ谷駅近辺の緯度経度
self.yotsuyaPoints = self.innerPathPoints(
                      139.7268,
                      35.6828,
                      139.7326,
                      35.6879,
                      proj="EPSG:2451"
                    )
```

次にエージェントの目的地を四ツ谷駅近辺に変更します。エージェントの目的地を変更する方法はいくつかありますが、ここでは setDestination() メソッドを使用する方法を説明します。

「NW エージェント」部品の「エージェントの初期化処理」において、setDestination() メソッドを呼び出します。引数は先ほど設定した四ツ谷駅近辺の経路地点リスト self.agentset.env.yotsuyaPoints からランダムに選択します。

エージェントの初期化処理

```
env = self.agentset.env
# 地点リストからランダムに 1 点サンプリング
yotsuya = next(sample(env.yotsuyaPoints))

self.setDestination(yotsuya)
```

この時点で実行ボタンを押すとシミュレーションが実行され、エージェントが信濃町駅近辺から出現して四ツ谷駅近辺へ移動する様子が確認できます。

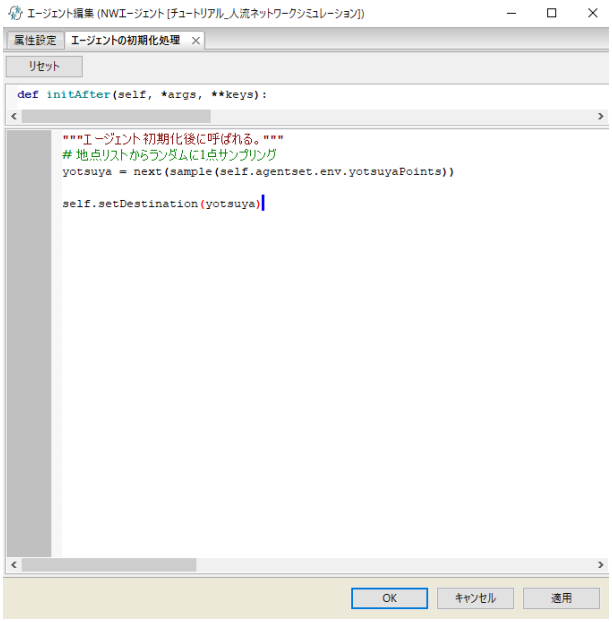


図 183: 目的地の変更

14.3.3 いくつかの経路地点を順番に移動させる

前節では信濃町駅を出発地として四ツ谷駅を目的地とした移動をシミュレーションしました。本節ではさらに目的地に行くまでにいくつかの地点を中継させる方法を説明します。

例として、信濃町駅から四谷駅に移動する途中で四谷三丁目駅を経由させることを考えます。

先ほどと同様に「NW 地図部品」の「環境の初期化後の処理」において、innerPathPoints() メソッドを呼び出して四ツ谷三丁目駅に最も近い経路地点の番号を取得し、変数 yotsuya3choumePoints としておきます。

環境の初期化後の処理

```
self.readOSM(
  "map.osm",
  139.7089, 35.6709, 139.7345, 35.6895
)

# (139.7202, 35.6800) は信濃町駅の経度、緯度
self.shinanomachi = self.nearestPathPoint(
                      (139.7202, 35.6800),
                      proj="EPSG:2451"
                    )
# (139.7268, 35.6828, 139.7326, 35.6879) は四ツ谷駅近辺の緯度
経度 (以下同様)
self.yotsuyaPoints = self.innerPathPoints(
                      139.7268,
                      35.6828,
                      139.7326,
                      35.6879,
                      proj="EPSG:2451"
                    )
self.yotsuya3choumePoints = self.innerPathPoints(
                              139.7185,
                              35.6856,
                              139.7234,
                              35.6897,
                              proj="EPSG:2451"
                            )
```

「NW エージェント」部品の「エージェントの初期化処理」において、エージェントを指定した経路地点を順番に移動させるメソッドである setTravel() メソッドを呼び出します。引数は四谷三丁目駅と四ツ谷駅の経路地点番号のリストです。

エージェントの初期化処理

```
env = self.agentset.env
# 地点リストからランダムに 1 点サンプリング
yotsuya = next(sample(env.yotsuyaPoints))
yotsuya3choume = next(sample(env.yotsuya3choumePoints))

# setDestination, は今回使用しないのでコメントアウトしておく。
# self.setDestination(yotsuya)
self.setTravel([yotsuya3choume, yotsuya])
```

この時点で実行ボタンを押すとシミュレーションが実行され、エージェントが信濃町駅近辺から出現して四谷三丁目駅を経由して四ツ谷駅近辺へ移動する様子が確認できます。

また、エージェントが到達した地点で指定した時間エージェントを滞留させたい場合、setTravel() メソッドのキーワード引数 stayTime を設定します。地点ごとに滞留時間を変えたい場合、キーワード引数 stayingKwargs を設定します。

エージェントの初期化処理

```
env = self.agentset.env
# 地点リストからランダムに 1 点サンプリング
yotsuya = next(sample(env.yotsuyaPoints))
yotsuya3choume = next(sample(env.yotsuya3choumePoints))

# 四谷三丁目駅近辺では 300 秒、四ツ谷駅近辺では 100 秒滞留する。
self.setTravel(
  [yotsuya3choume, yotsuya],
  stayTime=100,
  stayingKwargs={yotsuya3choume: {'t': 300}}
)
```

14.3.4 いくつかの地点をランダムに回遊させる

本節ではエージェントをいくつかの経路地点間をランダムに回遊させる方法を説明します。例として、信濃町駅、四ツ谷駅、四谷三丁目駅、千駄ヶ谷駅、青山一丁目駅をランダムに回遊させ続けます。

先ほどと同様に「NW 地図部品」の「環境の初期化後の処理」において、innerPathPoints() メソッドを呼び出して千駄ヶ谷駅近辺、青山一丁目駅近辺の経路地点番号のリストを取得し、変数 sendagayaPoints, aoyama1choumePoints としておきます。

環境の初期化後の処理

```
self.readOSM(
  "map.osm",
  139.7089, 35.6709, 139.7345, 35.6895
)

# (139.7202, 35.6800) は信濃町駅の経度、緯度
self.shinanomachi = self.nearestPathPoint(
                      (139.7202, 35.6800),
                      proj="EPSG:2451"
                    )
# (139.7268, 35.6828, 139.7326, 35.6879) は四ツ谷駅近辺の緯度
経度 (以下同様)
self.yotsuyaPoints = self.innerPathPoints(
                      139.7268,
                      35.6828,
                      139.7326,
                      35.6879,
                      proj="EPSG:2451"
                    )
self.yotsuya3choumePoints = self.innerPathPoints(
                              139.7185,
                              35.6856,
                              139.7234,
                              35.6897,
                              proj="EPSG:2451")
self.sendagayaPoints = self.innerPathPoints(
                        139.7100,
                        35.6796,
                        139.7133,
                        35.6819,
                        proj="EPSG:2451"
                      )
self.aoyama1choumePoints = self.innerPathPoints(
                            139.7226,
                            35.6716,
                            139.7259,
                            35.6739,
                            proj="EPSG:2451"
                          )
```

「NW エージェント」部品の「エージェントの初期化処理」において、エージェントを指定した経路地点間をランダムに回遊させるメソッドである setRandomTransition() メソッドを呼び出します。

setRandomTransition() メソッドを呼ぶ場合、ある地点から別の地点への遷移確率を引数として渡す必要があります。以下のコード例では変数 pattern として 2 地点間の遷移確率を与えています。

エージェントの初期化処理

```
env = self.agentset.env
# pattern のキーは経路地点番号型または経路地点番号型のタプルである必要がある。
shinanomachi = env.shinanomachi
yotsuya = tuple(env.yotsuyaPoints)
yotsuya3choume = tuple(env.yotsuya3choumePoints)
sendagaya = tuple(env.sendagayaPoints)
aoyama1choume = tuple(env.aoyama1choumePoints)

pattern = {
  shinanomachi: {
    yotsuya: 0.25,
    yotsuya3choume: 0.25,
    sendagaya: 0.25,
    aoyama1choume: 0.25
  },
  yotsuya: {
    yotsuya: 0.1,
    yotsuya3choume: 0.4,
    sendagaya: 0.1,
    aoyama1choume: 0.4
  },
  yotsuya3choume: {
    sendagaya: 1
  },
  sendagaya: {
    shinanomachi: 0.2,
    yotsuya: 0.1,
    yotsuya3choume: 0.3,
    aoyama1choume: 0.4
  },
  # 確率は重みづけでも指定可能
  aoyama1choume: {
    yotsuya: 2,
    yotsuya3choume: 1,
    sendagaya: 3
  }
}
# startPoint を指定することで回遊開始地点を信濃町駅に固定する
self.setRandomTransition(
  pattern,
  startPoint=shinanomachi
)
```

この時点で実行ボタンを押すとシミュレーションが実行され、エージェントが信濃町駅近辺から出現して設定した地点間をランダムに回遊する様子が確認できます。

setTravel() のときと同様に、エージェントが到達した地点で指定した時間エージェントを滞留させたい場合、setRandomTransition() メソッドのキーワード引数 stayTime を設定します。地点ごとに滞留時間を変えたい場合、キーワード引数 stayingKwargs を設定します。

エージェントの初期化処理

```
# 四ツ谷駅では 400 秒、それ以外では 200 秒滞留する例。
self.setRandomTransition(
  pattern,
  startPoint=shinanomachi,
  stayTime=200,
  stayingKwargs={yotsuya: {'t': 400}}
)
```

参考文献

[1] 北川源四郎, ' モンテカルロ・フィルタおよび平滑化について,' 統計数理, 1996, 44(1), 31-48.

[2] Geraerts, R.J.and Overmars, M.H. 'The Corridor Map Method: A General Framework for Real-Time High-Quality Path Planning' Computer Animation and Virtual Worlds, volume 18, pp. 107 - 119. 2007.

[3] D. Helbing and P. Molnar, 'Social Force Model for Pedestrian Dynamics', Physical Review E, 51, 5 1995.