### 引擎设计

许多现实世界的系统（如经济、生态、社会和城市系统）是高度复杂的。这些系统中的个体行为及其相互作用经常导致不可预测的宏观现象。传统的数学模型和仿真方法很难捕捉这种从微观到宏观的复杂动态。面对气候变化、疾病传播、经济危机等全球问题，我们需要更好地理解这些复杂系统的动态，以制定有效的策略和干预措施。目前的ABM建模工具要么专门针对某一应用领域，要么缺乏必要的灵活性和功能。这意味着研究者需要从头开始构建仿真，这既耗时又需要专业知识。为了让更多的研究者和决策者受益于Agent建模，需要开发一个用户友好、具有强大功能和广泛应用的工具。科学研究的一个关键方面是能够验证和复现结果。专业的Agent建模框架可以帮助确保模型的一致性和可靠性。复杂系统问题通常需要多学科的方法来解决。本平台可以作为不同领域专家之间的桥梁，帮助他们在一个共同的平台上合作，系统类图如图5-5所示。

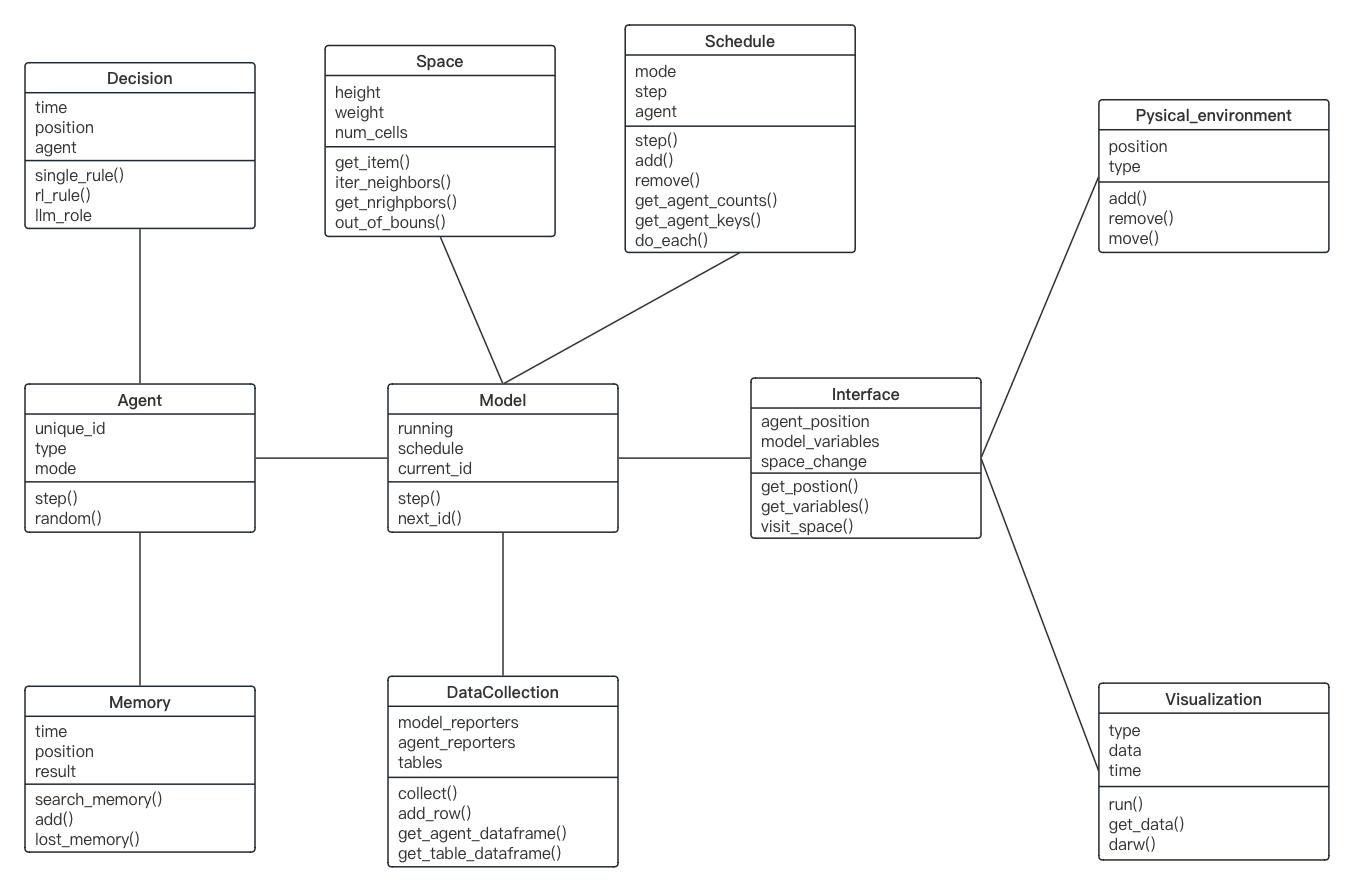


图5-5 系统类图

平台引擎作为开发的基础，主要包含以下几个模块，各个模块之间的关系见图5-5。Agent模块主要用于定义各类Agent；Model模块是Agent的容器，用于对Agent进行统一管理，一个Model模块中可包含多个Agent；Data Collection模块是引擎的数据收集模块，用于收集系统运行时产生的各类数据；Time模块是系统的时钟模块，用于对Agent的激活顺序进行控制；Batch Runner模块用于控制系统的批量运行，可以帮助用户更全面的理解模型和系统的行为；Environment模块是系统的环境模块，其中包含了各种环境的底层信息，方便建模者建立自己的环境模型。

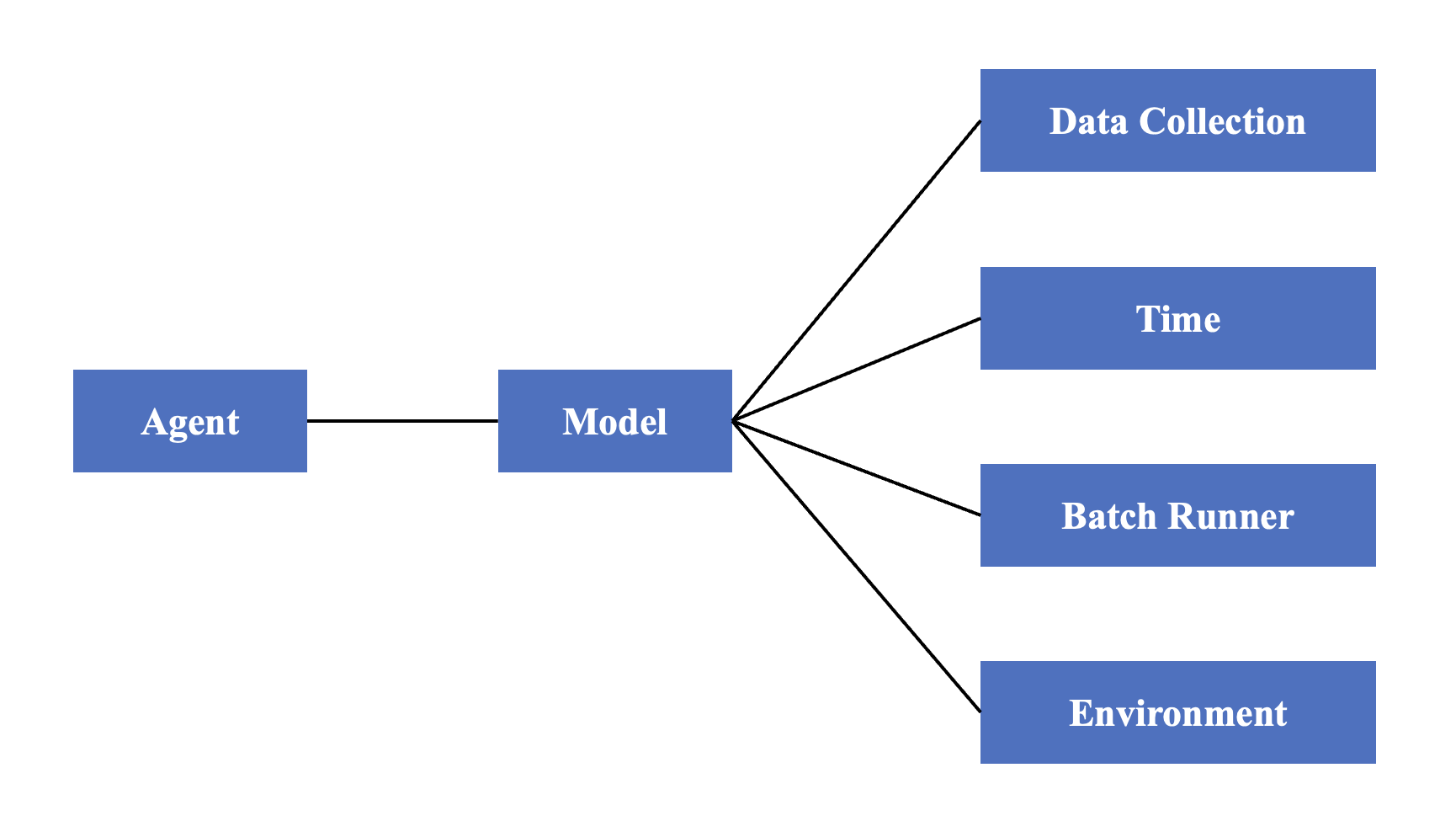


图5-6 系统引擎各模块及其之间的关系

#### **Agent**

多主体建模（Agent-Based-Modeling，ABM）中，Agent是模型的核心组成部分。Agent是具有一定属性、规则和行为的独立实体，它们可以与环境和其他Agent进行交互。在本系统中我们可以在模型中设置不同的智能体，并根据需要设定不同的角色，赋予其特定的认知能力、资源、属性以及判断流程和行动模式。

在许多复杂系统中，微观层面的个体行为和相互作用是导致宏观现象的关键，Agent为模拟这些个体行为提供了一个机制。不同的Agent可以具有不同的属性和行为，这可以反映真实世界中个体之间的差异。例如，在美团场景中，Agent可以是具有不同送餐能力的骑手。Agent可以根据一组定义的规则与其他Agent或环境进行交互，这有助于模拟例如竞争、合作、信息交流等复杂过程。Agent可以根据自己的经验或从环境中获得的信息来改变自己的行为。这种自适应性使得ABM能够模拟系统随时间演变的动态过程。Agent模型可以展示如何从单个Agent的简单规则出发，得到整体系统复杂的宏观行为。系统中每一个Agent基类设置的默认属性及方法如表5-1所示：

|  |  |  |
| --- | --- | --- |
|  | 属性/方法名 | 说明 |
| 属性 | unique\_id | Agent独一无二的id，方便在后续进行管理 |
| pos | Agent当前所处的位置 |
| model | 对agent进行管理的容器 |
| 方法 | step | 每一个时间步agent所采取的行动，默认方法为空，可重载 |
| random | 随机方法，获取模型的激活顺序 |

表5-1 Agent模块默认属性及方法

#### **Model**

Model是多主体建模中的模型，在本系统中主要的作用就是容纳并管理系统中的多个Agent，并对其进行调度。Model充当对现实世界问题或系统的抽象表达，为其结构和动态提供了框架。这个模型明确了Agent的行为准则和与环境的交互方式。Model还定义了Agent所处的环境，这可能包括空间的表示（如二维格网或连续空间）、资源的分布、环境的规则等。Model为系统中的相互作用和流程定义了特定的规则和约束，从而确保模拟过程与真实世界的情况相一致。Model不仅仅是一个静态的结构；它通过时间的进程模拟了系统的动态演变。这意味着Agent和环境状态的更新、决策的实施和Agent之间的互动都是模型运行的一部分。一旦构建完成，Model成为一个实验的平台，允许研究者修改参数、初始条件或Agent规则来探索不同的情景或假设。Model可以基于理论、观察或实验数据构建，从而为真实世界的现象提供解释或预测。通过Model，研究者可以在一个受控的环境中探索这个系统的行为、演变和潜在的干预效果。系统中为每一个Model设置的默认属性如表5-2所示。

|  |  |  |
| --- | --- | --- |
|  | 属性/方法名 | 说明 |
| 属性 | running | 标记模型的运行状态 |
| schedule | 模型运行所遵循的时间表，即模型中所容纳的Agent的激活顺序（顺序激活、随机激活、按照优先级激活等） |
| current\_id | 目前循环进行到的Agent的id |
| 方法 | run\_model | 判断模型运行是否达到终止条件，如果是则终止运行，否则继续运行 |
| next\_id | 下一次循环的Agent的id |
| initialize\_data\_collector | 对数据收集的配置进行初始化 |
| step | 每个循环步骤下Model所采取的措施，默认方法为空，可重载 |

表5-2 Model模块默认属性及方法

#### **Environment**

在ABM建模中，环境是非常重要的一个组成部分。Environment是Agent行动和交互的背景或舞台。它不仅为Agent提供了空间和上下文，而且可能包含重要的资源、约束和其他外部影响因素。Environment为Agent提供了一个空间框架，Agent可以在其中移动、互动或寻找资源。系统不仅提供了一个简化的二维格网，也提供了更复杂的连续空间和网络结构。许多模型中的Agent需要消耗、获取或争夺环境中的资源，如食物、土地或资金。Environment定义了这些资源的类型、数量和分布。Environment可能包括各种外部因素，如天气、政策或市场条件，这些因素都会对Agent的决策和行为产生影响。Agent的行为可以改变环境的状态，而这种变化又可能反过来影响Agent。例如，系统中骑手的数量过多可能会导致骑手的收益下降，从而导致骑手数量变少，而骑手数量减少又会反过来会影响剩余骑手的收益。Environment可能包含Agent需要的信息，这有助于Agent做出决策。例如，骑手会根据自己观察到的现象来判断哪里订单数量多，从而决定在哪个区域送餐。

在ABM中，Environment为Agent的行动和决策提供了上下文，同时它也是模型中的一个动态成分，可以受到Agent行为的影响并反过来影响Agent。这种Agent与环境之间的双向交互是ABM的核心特征之一，它允许模型捕捉复杂系统中的反馈循环和自适应行为。系统中主要设置了下列几种底层环境，分别是矩形网格环境（一个网格只允许容纳一个Agent、一个网格可容纳多个Agent）、六边形网格环境一个网格只允许容纳一个Agent、一个网格可容纳多个Agent）、连续空间环境，如表5-3所示。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 矩形网格 | 六边形网格 | 连续空间 |
| 容量 | 一个网格仅容纳一个Agent | 一个网格仅容纳一个Agent | / |
| 一个网格可容纳多个Agent | 一个网格可容纳多个Agent |

表5-3 Environment模块物理环境

网格环境将空间划分为规则的格子，Agent在这些格子中移动和互动。网格环境简单易于实现，对于某些问题（如土地使用或空间扩散）特别适用，并且方便进行邻近性分析，因为邻居的定义很明确（例如，上下左右或对角线格子）。系统网格环境的默认属性和方法如表5-4所示。

|  |  |  |
| --- | --- | --- |
|  | 属性/方法名 | 说明 |
| 属性 | width | 网格的宽度 |
| height | 网格的高度 |
| 方法 | is\_cell\_empty | 判断网格中当前单元是否为空，主要适用于单个网格单元仅能容纳一个Agent的情况 |
| get\_item | 获取网格中的内容，如果网格中没有内容则返回空，如果网格中有一个Agent，则返回该Agent的信息，如果网格中有多个Agent，则返回Agent信息列表 |
| out\_of\_bounds | 断坐标是否脱离网格范围 |
| iter\_neighborhood | 通过迭代的方式获取指定网格的邻居，可以通过指定获得冯诺依曼邻居或者摩尔邻居 |
| get\_neighborhood | 通过返回列表的方式获取指定网格的邻居，可以通过指定获得冯诺依曼邻居或者摩尔邻居 |
| place\_agent | 把Agent放置到指定的坐标点上 |
| remove\_agent | 把Agent从指定坐标点上移出 |
| move\_agent | 把Agent从当前位置移动到指定位置 |
| swap\_pos | 交换两个Agent的位置 |
| move\_to\_empty | 把Agent随机移动到一个空单元 |
| exists\_empty\_cells | 返回网格中是否还有空单元 |

表5-4 网格环境默认属性及方法

与网格空间有所不同的是连续空间，在网格空间中Agent的坐标都是整数，而在连续空间中Agent的坐标可以是浮点数。在这种环境中，空间不是离散的，而是连续的。在连续空间中，Agent的移动灵活性比网格空间更大。更为真实地表示物理空间。并且允许更细致的空间交互和移动模式，不受格子大小的约束，从而避免了某些伪像问题。在本系统中，连续空间中的一些属性和方法继承自网格空间，同时其也具有自己特殊的属性和方法，连续空间独有的默认属性及方法如表5-5所示。

|  |  |  |
| --- | --- | --- |
|  | 属性/方法名 | 说明 |
| 属性 | x\_min | 连续空间x坐标的最小值 |
| x\_max | 连续空间x坐标的最大值 |
| y\_min | 连续空间y坐标的最小值 |
| y\_max | 连续空间y坐标的最大值 |
| \_index\_to\_agent | 索引值与Agent对应的字典 |
| \_agent\_to\_index | Agent与索引值对应的字典 |
| 方法 | get\_distance | 获得两点之间的最短距离 |

表5-5 连续环境独有的默认属性及方法

#### **Time**

Agent的激活顺序描述了在每一个模拟的时间步长中，Agent的行为和决策是如何按照特定的顺序执行的。Agent的激活顺序可能会对模型的输出产生重大影响，因此选择合适的激活策略是模型设计中的一个关键考虑因素。不同的激活顺序可能会导致模型的不同输出。例如，在派单算法中，骑手的不同激活顺序可能导致先激活的骑手比后激活的骑手有更多的机会获得订单。不同的激活顺序可能会导致模型的不同输出。例如，在一个资源争夺的场景中，先行激活的可能比后续的Agent有更多的机会获得资源。在某些情境中，某种特定的激活顺序可能更贴近真实世界的动态。例如，在美团场景中，可能需要商家在骑手之前被激活。允许动态或随机的激活顺序可能会增加模型的复杂性和不确定性，但同时也可能增加模型的真实性和鲁棒性。本系统提供五种激活顺序：Base Schedule（按照Agent添加的顺序激活）、Random Activation（随机激活）、Simultaneous Activation（并行激活）、Staged Activation（按照所设定的Agent的等级激活）、Random Activation By Type（按照Agent的类型随机激活）。在本系统中，基本时间表具有的默认属性及方法如表5-6所示。

|  |  |  |
| --- | --- | --- |
|  | 属性/方法名 | 说明 |
| 属性 | model | 时间表所控制的模型 |
| steps | 当前进行到的时间步 |
| \_agents | 时间表所控制的Agent字典 |
| 方法 | add | 在时间表中添加Agent |
| remove | 将Agent从时间表中移除 |
| step | 执行所有Agent的step方法 |
| get\_agent\_count | 获取当前时间表中所具有的Agent数量 |

表5-5 Time模块默认属性及方法

#### **DataCollection**

在模型运行过程中，本系统还提供了数据收集方法。数据收集模块是非常重要的，因为它允许研究者从模拟中捕获和分析信息。数据收集模块允许研究者捕获模型的输出，这些输出可能包括代理的状态、行为、交互以及整个系统的全局属性。通过比较模型的输出与实际观察数据或预期结果，研究者可以验证和校准模型。数据收集模块为这种比较提供了必要的数据。通过数据收集，研究者可以对模型输出进行统计分析，生成图表、地图和其他可视化工具，从而更好地理解和解释模型的行为。数据收集模块在ABM中扮演了核心角色，它为模型验证、分析和解释提供了必要的数据和工具。数据收集主要分为三种类型的数据：Agent数据（Agent运行过程中的属性数据）、模型级数据（模型运行过程中产生的数据）、表格数据（自定义的表格数据）。系统中数据收集类的默认属性如下，默认方法如表5-6所示。

* model\_reporters：模型级数据
* agent\_reporters：Agent级数据
* tables：自定义表格

|  |  |
| --- | --- |
| 方法名 | 说明 |
| \_new\_model\_reporter | 在记录中添加一个需要收集的模型数据变量 |
| \_new\_agent\_reporter | 在记录中添加一个需要收集的Agent数据变量 |
| \_new\_table | 在记录中添加一个需要收集的表格 |
| \_record\_agents | 记录Agent信息 |
| collect | 收集数据 |
| get\_model\_vars\_dataframe | 返回收集到的模型数据的DataFrame |
| get\_model\_vars\_dataframe | 返回收集到的Agent数据的DataFrame |
| get\_table\_dataframe | 返回收集到的表格数据的DataFrame |

表5-6 Data collection模块默认方法

#### **BatchRunner**

在使用计算实验进行模拟仿真时，一次实验结果可能具有偶然性，我们需要多次实验来观察普遍规律，同时我们也希望采用不同的配置来运行同一个实验，期望获得不同的结果。Batch Runner是一个用于批量运行模型的工具或模块。它允许研究者自动化地进行多次模拟运行，每次运行都可能使用不同的参数组合或初始条件。

Batch Runner允许研究者在一个参数的范围内运行多次模拟，以便更好地了解参数变化对模型输出的影响。通过使用Batch Runner进行大量的模拟运行，研究者可以评估模型参数对输出的灵敏度，从而识别那些对模型结果影响最大的参数。因为ABM经常具有随机性，单次模拟的结果可能会有很大的变异性。通过进行多次模拟，研究者可以得到模型输出的平均值、分布和统计特性，从而提供更鲁棒的结论。Batch Runner使研究者能够在多种条件下探索模型的行为，从而更好地了解和解释模型在不同场景下的反应。手动进行大量的模拟运行是非常耗时的。该模块提供了自动化的解决方案，使研究者可以高效地完成这些运行。并且通常与数据收集模块紧密集成，允许研究者为每次模拟运行自动收集和存储数据。在批量运行函数中主要传输的参数如表5-7所示。

|  |  |
| --- | --- |
| 参数名 | 说明 |
| model\_cls | 需要批量运行的模型类 |
| parameters | 批量运行的模型参数 |
| process | 进程数量，默认为1 |
| iterations | 变量的迭代次数 |
| data\_collection\_period | 模型运行多少步之后开始收集数据，默认值为-1 |
| max\_steps | 模型运行的最大时间步 |

表5-7 Batch Runner传入参数