Menge: A Modular Framework for Simulating Crowd Movement

Sean Curtis[1] · **Andrew Best**[1] · **Dinesh Manocha**[1]

[1] University of North Carolina at Chapel Hill E-mail: seanc,best,dm@cs.unc.edu

Received: date1 / Accepted: date2

DOI `XX.XXXX/XXX` / Online at `collective-dynamics.eu`

Abstract We present Menge, a cross-platform, extensible, modular framework for simulating pedestrian movement in a crowd. Menge's architecture is inspired by an implicit decomposition of the problem of simulating crowds into component subproblems. These subproblems can typically be solved in many ways; different combinations of subproblem solutions yield crowd simulators with likewise varying properties. Menge creates abstractions for those subproblems and provides a plug-in architecture so that a novel simulator can be dynamically configured by connecting built-in and bespoke implementations of solutions to the various subproblems. Use of this type of framework could facilitate crowd simulation research, evaluation, and applications by reducing the cost of entering the domain, facilitating collaboration, and making comparisons between algorithms simpler. We show how the Menge framework is compatible with many prior models and algorithms used in crowd simulation and illustrate its flexibility via a varied set of scenarios and applications.

Keywords Crowd Simulation · Open Source · Software System

1. Introduction

Whether for interactive graphics, special effects, or engineering applications, crowd simulation – the simulation of a large number of independent entities acting and moving through a shared space – relies on the solution to many subproblems: determining what an agent wants to do, how it will achieve its purpose, how it responds to unforeseen challenges, and, for visual applications, determining how its virtual body moves. These subproblems are manifest in computer graphics, robotics, animation, psychology, pedestrian dynamics, and biomechanics literature, where significant work has been performed to provide increasingly superior solutions. A full crowd simulator can be regarded as the union of solutions to each of these subproblems.

Each of these subproblems typically admits various solutions. For example, the problem of determining how an agent reaches its goal can be mapped to global motion planning. To solve this subproblem, one could use algorithms including, but not limited to, potential fields [31], road maps [37], navigation meshes [58], or corridor maps [14]. Selecting one is a non-trivial choice. First, each of these approaches has its own strengths and weaknesses – there are some problem domains for which a particular approach may be better suited than others. Second, implementing one approach may be more complex than another. Third, while each approach will solve the subproblem, the solutions may not be the same; the choice of how a subproblem is solved can have an impact on the resulting agent behavior.

The inherent complexity of creating a functional crowd simulator can also serve as an obstacle to researchers and developers. Developing a full system is complex and time consuming. Even if a researcher is interested in a single aspect of crowd simulation, proper evaluation of a novel technique requires the greater context of a full simulation system. Every researcher who implements an *ad hoc* crowd simulator, for the express intent of testing one component, spends time and effort only tangentially related to their core research. Worse yet, this effort is duplicated across independent research groups.

In addition, each time an entire crowd movement simulator is created to support the creation of a single component, the task of performing meaningful comparisons between novel and pre-existing approaches becomes increasingly difficult. Currently, the best common practice is a straightforward implementation of published models for comparison. But in these cases, a reimplementation of a paper is unlikely to be the same as the author's original, rendering the significance of the comparison uncertain.

Research in and development of crowd simulation applications would benefit from a common framework. This common framework would be architected with a view of the various subproblems in mind; each subproblem would be encapsulated within an appropriate interface. Novel solutions to subproblems could be incorporated with other solutions drawn from a library. A common framework would contribute to the science of crowd research, not through novel models or algorithms, but by facilitating subsequent research. We posit that such a framework would have multiple important benefits:

- Low-cost entry: Researchers would not be obliged to create a simulator from scratch. Researchers first entering the domain could focus on one aspect, but still evaluate it in a complex context by exploiting the frameworks built-in implementations.
- Focused development: Researchers could focus on a single subproblem, while exploiting shared implementations of solutions for the surrounding context. This would reduce the initial cost of performing research in crowd simulation.
- Efficient dissemination: Novel solutions to subproblems could be released (either in code or in binary form) into the common framework, allowing other users to make use of the novel models, exploiting their improved properties.
- Meaningful comparison: As users release their novel subproblem solutions back to the framework, other users of the framework could make direct, meaningful com-

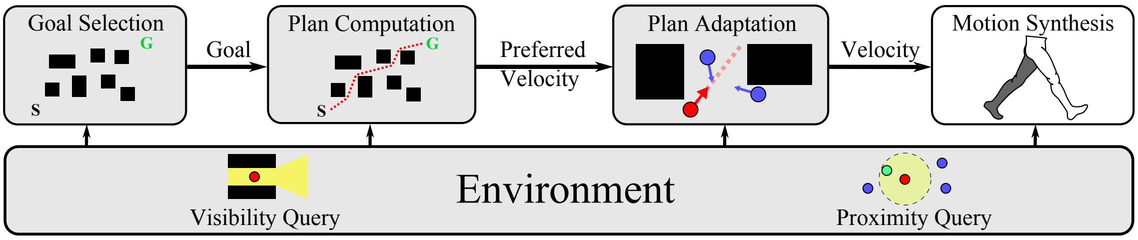


Figure 1 An abstraction of crowd simulation based on subproblems. First, a goal is selected. Second, a base plan to reach that goal is computed. Third, the plan is adapted to local, dynamic conditions. Finally, motion is synthesized in support of the realized plan. Each subproblem can make queries into the environment to support its computation. Only those elements in grey are included in Menge, although Menge is capable of propagating complex agent state to the motion synthesis stage.

parisons with previous results because they are running the original implementation in its original context.

- Bespoke functionality: Custom components could be introduced according to the needs of a particular simulation problem.

- Flexible specification: In order to simulate varied, complex, real-world scenarios, the framework would be able to define simulation scenarios efficiently.

To that end, we present Menge, a modular, open-source, cross-platform framework for simulating crowd movement, explicitly designed to realize all of the desired beneficial properties outlined above. Moreover, we argue that Menge's ability to provide this broad set of benefits is unique among the various simulation applications which have been released by the crowd simulation community.

The discussion in this paper provides the evidence in support of this position – that Menge's properties make it uniquely capable of providing the benefits outlined above. We discuss Menge's underlying paradigm and show that many broad categories of crowd research work implicitly fit this paradigm in Section 2. In Section 3, we present the architecture designed to realized the targeted properties. We provide examples of Menge applied to meaningful research problems in Section 4, illustrating the research benefits of the architecture. Section 5 summarizes Menge's unique capacity to serve as a common simulation framework by comparing and contrasting its benefits with those of other, publicly available crowd simulation systems. Finally, we offer our concluding thoughts in Section 6.

2. Simulating Crowds

Menge realizes a particular abstraction of crowd movement simulation. The abstraction is a decomposition of the problem into related *subproblems*: goal selection, plan computation, plan adaptation, and spatial queries (see Figure 1). This is not a novel abstraction;

it has been referred to in previous work [13,61] and is well represented in the crowd simulation literature. In this section, we discuss representative work in crowd simulation in the context of this abstraction.

2.1. Goal Selection

The first subproblem, *goal selection*, involves determining what each pedestrian wants to achieve. Generally, decisions of this type can incorporate diverse factors, e.g. psychology, world knowledge, etc. What the pedestrian wants to achieve can change with respect to time and conditions. The complexity required depends on the simulation scenario and can range from simple (flow down a corridor) to complex (populating a train station).

The problem of determining what an agent *wants* to do has been extensively explored. Shao and Terzopoulos [54] used *situation calculus* to author a complex train platform scenario . Ulincy and Thalmann [61] computed high-level behaviors with a combination of rules and behavior finite state machines. Similarly, Bandini et al. [1] used finite state machines to model complex behaviors with a cellular automata pedestrian model. Paris and Donikian use a hierarchical finite state machine to determine high-level agent behaviors (although it is used to determine sub-tasks selected to reach the pre-defined, ultimate goal) [46]. Generally, this domain is solved using some form of decision or network graph. The product of this stage, a “goal”, is provided to the next stage as input.

2.2. Plan Computation

The second subproblem, *plan computation*, seeks to create a *static* plan to achieve the goal. This is most typically associated with *motion planning* [37]. In crowd simulation, if the goal requires the agent to perform an action at its current location, the motion planning consists of motion synthesis of the pedestrian’s visual representation[1]. If the goal requires the agent to traverse the simulation domain, then the problem combines “path planning” *and* motion synthesis. The path is an abstract concept. An agent’s path defines an agent’s *preferred velocity* – the velocity the agent would take at any given moment to make progress towards its goal.

There are multiple approaches for computing paths. Many of them are predicated on discretizing the traversable space into connected primitives. The connected primitives imply a graph which can be searched using standard algorithms (e.g., A*). These graph-based algorithms include: road maps [37], navigation meshes [44,58], delaunay triangulation [36], and corridor maps [14]. These data structures have traditionally been applied to traversable space with respect to static obstacles, but work has also been performed to adapt them to dynamic changes to traversable space (e.g., [25,27,63]).

Another common approach uses potential fields. The simulation domain is discretized and a field is computed that is the gradient of a cost function [31,35]. No path is explicitly computed. Instead, the resultant vector field provides a direction of “optimal” travel

---

[1] As indicated in Figure 1, Menge does not include motion synthesis, but its simulation output is compatible with off-line synthesis.

toward the goal. Plan computation's ultimate product, preferred velocity, serves as input to the next subproblem.

2.3. Plan Adaptation

Typically, computed plans only consider static obstacles and low-frequency phenomena. This gives rise to the third subproblem, *plan adaptation*. Rather than recomputing a plan each time the simulation environment changes, the plan is adapted to handle local, dynamic obstructions as needed. This subproblem has many names: "pedestrian model", "local navigation", "steering", etc. Essentially, the solution to this subproblem transforms the ideal, preferred velocity into a *feasible* velocity.

There are a large number of models which adhere to this paradigm. Such approaches include, cellular automata [53], social forces [22, 30, 47], vision-based [45], continuum-based [42], velocity-obstacle-based [3, 48], and rule based [50]. *All* of these models are compatible with the plan adaptation abstraction. This list is meant to be representative of classes of simulation paradigms; for a more thorough discussion, please see contemporary surveys [11, 65].

It is worth noting that there are crowd models which use a different paradigm (e.g., [29, 60]). In these problems, the plan computation and adaptation are collapsed into a single problem; the plan computation considers the full domain, rendering adaptation largely unnecessary. Even with these differences, they could still be implemented in Menge; in this case, all of the work would be performed during plan computation, and the plan adaptation would be an identity operation.

2.4. Motion Synthesis

For visual applications, it is necessary to compute physical character motion consistent with the activity computed by the previous stages. There has been a great deal of work in this field including procedural methods [6, 59], data-driven methods [20, 34, 38, 41], and, for locomotion, foot-step driven methods [2]. In its current release, Menge does not directly address this issue[2].

2.5. Environmental Queries

Finally, the various subproblems typically need to perform spatial queries in the environment. For example, it is reasonable to limit the effect of the environment on an agent to those factors which are in the line of sight to the agent (visible) or near the agent (proximal). To support this type of operation, we require the ability to perform spatial queries such as visibility queries or proximity queries. For details on the many solutions to these types problems, we refer the reader to the following resources for visibility queries [9] and proximity queries [52].

---

[2] The visualizations shown in Section 4 have been produced by a proprietary visualizer using Menge's output data.

2.6. Crowd Systems

There is also research in full crowd simulation systems. Autonomous Pedestrians, in part inspired by Newell's [43] Unified Theories of Cognition, expresses the crowd simulation problem as a composition of conceptual layers [54]. These conceptual layers correspond well to Menge's abstraction of goal selection, plan computation, and plan adaptation. Other open-source simulation systems have been released, e.g., SteerSuite [56], ADAPT [28], etc. We provide a detailed comparison with these systems in Section 5.

3. Menge's Architecture

In this section, we discuss the design philosophy and architecture of Menge. We analyze how this architecture realizes the benefits of a common simulation framework in Section 5.

3.1. Mathematical Realization

Menge's architecture is primarily focused on facilitating the simulation of agents *moving* through a shared space.[3] The problem of computing agent trajectories can be thought of as an initial value problem (IVP):

$$\dot{\mathbf{x}}_i(t) = \mathbf{v}_i(t) = \mathbf{V}_i(t, \mathbb{S}(t)), \tag{1}$$

where $\dot{\mathbf{x}}_i(t)$ or $\mathbf{v}_i(t)$ is the instantaneous velocity of agent i at time t, $\mathbb{S}(t)$ is the simulator *state*, likewise at time t, and $\mathbf{V}_i$ is a function that determines the agent's instantaneous velocity. By solving for $\mathbf{x}_i(t)$, we determine the position of the agent with respect to time.

The simulator state $\mathbb{S}$ is the union of all entities in the scene, including the features of the simulation domain (e.g., obstacles) and the full crowd state space. The crowd state space $\mathbb{X} = \bigcup_i \mathscr{X}_i$ is the union of each agent's state space. The minimum agent state space necessary to satisfy the differential equation is $\mathscr{X}_i = [\mathbf{x}_i \; \mathbf{v}_i]^T$, where $\mathbf{x}_i$ and $\mathbf{v}_i \in \mathbb{R}^2$. Menge assumes that simulation is performed in a two-dimensional domain[4]. In practice, particular solutions to the initial value problem require additional per-agent properties which extend the agent state.

Ultimately, the properties of the crowd simulator, and the behaviors its agents exhibit, is dominated by the agent state and, more particularly, the velocity function $\mathbf{V}_i$.

3.2. Conceptual Abstraction as Functions

We can easily map each of the conceptual subproblems into functions. Furthermore, we can compose those functions to define the velocity function $\mathbf{V}$. The IVP abstraction may

---

[3] Menge's architecture can also account for simulation in which agents remain stationary but nevertheless have changing relationships with respect to each other and their environment (see Section 3.3.)

[4] Although allowances are made for three-dimensional simulation domains that are only *locally* two-dimensional.

admit other mappings, but this mapping supports the modular formulation which is one of Menge's design goals.

The goal selection subproblem would be: $G_i : t \times \mathbb{S} \rightarrow \mathbb{R}^2$. For a single agent i, this function maps time (t) and simulation state ($\mathbb{S}$) into a two-dimensional goal position[5].

The plan computation becomes *path* computation and its corresponding function, $P_i : t \times \mathbb{S} \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$, maps time, simulation state, and the agent's goal position into an instantaneous preferred velocity.

Finally, the plan adaptation function, $A_i : S_i \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$, maps the preferred velocity and *local* simulation state into a feasible velocity. Generally, the adaptations are assumed to have limited temporal validity, so in this case, "local simulation state" refers to the simulation features near the agent i.

The simulation state serves as a parameter to all three functions. By assuming that implicitly, the functions simplify to: $G_i : t \rightarrow \mathbb{R}^2$, $P_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, and $A_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$. The instantaneous velocity of an agent is the composition of these functions: $\mathbf{V}_i(t) = A_i(P_i(G_i(t)))$ and can be substituted into Eq. 1 as:

$$\mathbf{v}_i(t) = A_i(P_i(G_i(t))). \qquad (2)$$

Menge implements this abstraction. Each subproblem function is implemented by a set of one or more orthogonal *elements*. A particular crowd simulator can be instantiated by specifying particular elements and their relationships. For example, configuring two different simulators such that they use the same solutions to the goal selection and path planning subproblems, but different path adaptation solutions is trivial; one simply changes the reference to the path adaptation module in the Menge project file (see Section 4.1 for specific examples). This is how one would perform comparisons between two or more steering algorithms.

3.3. Stationary Agents

Any crowd simulation system which is primarily focused on *moving* agents would seem to inherently consider all stationary agents to be equivalent. In reality, two stationary agents could still have significantly different properties, goals, and relationships with their surroundings. Menge's architecture makes it possible to distinguish between two agents which may otherwise have identical trajectories (e.g., standing still) via its Behavioral Finite State Machine (BFSM). Two stationary agents could occupy different states in the BFSM, representing different activities or mental conditions. The trade show example in Section 4.1 illustrates just this distinction.

3.4. Architectural Elements

Menge's modular architecture is based on the concept of *elements*. An element type defines a particular aspect of a subproblem. The element type defines an interface that

---

[5] A goal point in $\mathbb{R}^2$ is a common simplification; goals could be regions. But for many applications, this simplification is sufficient.

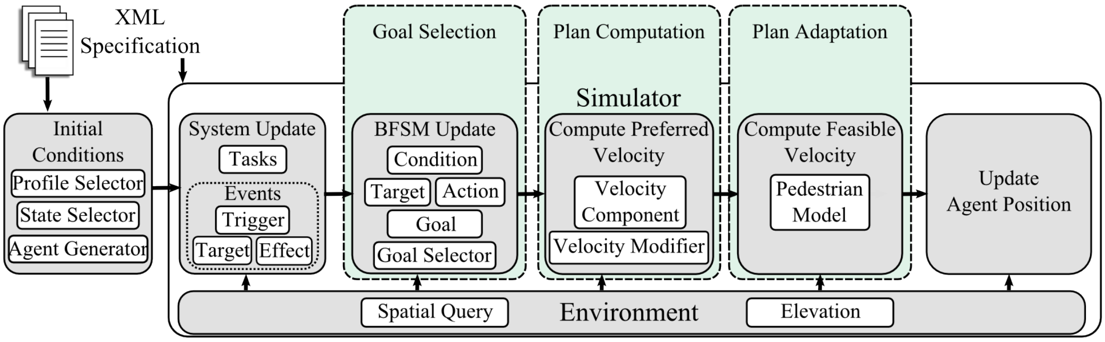


Figure 2 Menge's computation pipeline. The modular *elements* are shown in white boxes. The green boxes show how the elements relate to the conceptual subproblems. The simulator definition (including initial conditions and BFSM) is given as an XML specification. At each time step, the system updates event state and task state. Then the BFSM is updated for each agent. Next, the preferred velocity for each agent is computed. The pedestrian model is used to compute a feasible velocity. Finally, the agent position is updated.

can be implemented to provide a particular solution. Each element type can have an arbitrary set of implementations. The implemented elements are explicitly instantiated via the XML specification. In its initial release, Menge includes a set of representative implementations for each element type.

The elements are grouped by functional purpose. Conceptually, we ascribe selecting a goal and planning a path to an agent's *behavior*. We model agent behavior and how it changes with respect to time with a Behavioral Finite State Machine (BFSM). As such, the *Goal Selection* and *Plan Computation* problems are solved by elements which belong to the BFSM. The *Plan Adaptation* domain belongs to the pedestrian models element. Menge's underlying system is exposed via the system elements and, finally, initial scenario conditions are defined by a set of appropriate elements. We will now discuss the seventeen elements which make up Menge's modular architecture, as illustrated in Figure 2. An example scenario specification can be seen in the Appendix.

Agent state: We have previously introduced the agent state as the vector $[\mathbf{x}_i\ \mathbf{v}_i]^T$. These properties are sufficient to express the initial value problem, but in practice, for a particular agent model, more parameters are required. We refer to the agent state vector consisting of position and velocity as the *agent state*, or a-state. The set of additional properties (e.g., radius of disk, response time, etc.) will be called the *behavioral state* or b-state; modifying these properties changes how the agent's trajectory is computed, leading to a different agent behavior. Finally, to avoid confusion, we will refer to a state in the BFSM as an agent's FSM-state.

3.5. Behavioral Finite State Machine Elements

The Behavioral Finite State Machine encapsulates the core of agent behaviors. Each state in the BFSM governs what goal the agent seeks, how it intends to achieve that goal, and can even influence the agent's fundamental characteristics, modeling changes in mood and thought. The transitions from one state to another govern changes in the agent's behavior.

Finite state machines have been shown to be quite effective for this purpose [1,61]. The specification of a particular BFSM defines how the agents interact with their environment and each other and how those relationships change with time.

`Condition`

The `Condition` element, in conjunction with the `Target` element, defines a BFSM transition. The `Condition` provides a boolean test which determines if a pre-defined condition is satisfied. If so, the transition is activated and the agent exits its current FSM-state and moves to the FSM-state defined by the transition's `Target` element. An FSM-state can be connected to multiple out-going transitions. These transitions are prioritized and the condition of each transition is evaluated in priority order; the first transition whose condition is met is taken.

The `Condition`'s boolean test can consist of arbitrary logic. Menge's default implementation contains implementations which depend on temporal, spatial, and stochastic parameters. For example, in simulating passengers disembarking an airplane, an agent might wait to leave its seat until the aisle is empty; this would be realized with a custom `Condition`. Pre-existing conditions can be combined or new implementations can be introduced to the system via its plug-in architecture to achieve desired results. See Section 4.2 for examples.

`Target`

The `Target` element determines which FSM-state an agent moves to when the corresponding `Condition` is satisfied. In a strictly-defined finite state machine, a transition would connect one source FSM-state to one destination FSM-state. When defining agent behavior via the BFSM, it can be convenient to model the behavior that a single condition could lead to one of a set of new FSM-states, based on some additional criteria. The `Target` element makes this possible in a compact manner. Menge includes targets which allow transitions to a single FSM-state, transition to a randomly selected member of a set of FSM-states, or an automatic return to the FSM-state preceding the current state. Simulating a train station would provide a simple example; following a "buying-ticket" FSM-state an agent might proceed to concessions or their train platform. The probabilistic target will allow for a controlled distribution of behaviors. As with all elements, new target implementations are easily introduced.

`Action`

The `Action` element allows an FSM-state to directly make changes to an agent's a-state or b-state; the BFSM *acts* on the agent and not, as the name may suggest, an action taken *by* the agent. `Actions` are executed on an agent when the agent enters the FSM-state and can be configured to undo the change when the agent leaves the FSM-state or not, as appropriate for the simulation. These actions can be used to varying effect. For example, stress can be modeled by an agent successively entering an "increased stress" FSM-state where each time, an `Action` modifies the agent's b-state properties to represent a heightened response to stress (see Section 4.2). An `Action` element can also be used for reasons of convenience. For example, a simple scenario with periodic boundaries can be simulated by including an `Action` which teleports agents from their current position back to the beginning of a straight hallway (demonstrated in the Appendix).

3.6. BFSM Goal Selection Elements

In simple scenarios, goal selection can be defined externally to the simulator and remain constant for the simulation duration (e.g., flow down a corridor). In complex scenarios, the agent's goal can change from moment to moment. These changing goals are modeled using the FSM-states. Upon entering an FSM-state, an agent is assigned a `Goal` using the `Goal Selector` element associated with that FSM-state.

`Goal`

The `Goal` element is the basic primitive for defining the space the agent wants to reach. As previously indicated in Section 3.2, an agent's goal is a region in two-dimensional space. Menge's default implementation contains a number of simple, convex regions (a point, a circle, an axis-aligned box, and an oriented box). At any given moment, the agents seek to move toward the nearest point in the region. By defining `Goals` as two-dimensional regions, `Goals` can be efficiently shared by multiple agents without causing artificial queuing arising from agents waiting to access, what would otherwise be, a point goal. Regions inherently have a greater capacity to accommodate multiple agents. Menge `Goals` can have finite "capacity", meaning that there is a limit on the number of agents which can simultaneously share that goal. Each agent pursuing that goal consumes a portion of the capacity; when all capacity is taken, no more agents can be assigned that goal.

`Goal Selector`

The `Goal Selector` element is the primitive which defines the basis for assigning an agent a `Goal`. When an agent enters a state, its `Goal Selector` is evaluated and a `Goal` is assigned to the agent. When the agent leaves the FSM-state, the goal is "released". This behavior is configurable; the `Goal Selector` can be made "persistent", meaning that the `Goal` assigned the first time the `Goal Selector` is evaluated is not freed up when the agent leaves the state. This allows the agent to return to the state and return to its original goal. It also means that the capacity of that goal is not freed up. Furthermore, this persistent goal can be shared across multiple states via "goal sharing." Menge includes a wide range of goal selectors including: a single, pre-defined `Goal`, a uniform or weighted random selection from a set, the nearest or farthest to the agent's current position in the set (based on Euclidian distance), the nearest or farthest based on path length through a navigation mesh, and more. Ultimately, a novel `Goal Selector` could include arbitrary algorithms for selecting a `Goal`. For example, pedestrian simulation was used in the redesign of the London Bridge Station. Surveys of passenger behaviors were used to build a statistical model for assigning destinations [23]. This statistical model could serve as stochastic weights on a set of `Goals`.

3.7. BFSM Plan Computation Elements

Solutions to the plan computation subproblem must provide an instantaneous *preferred velocity*; at any given time, an agent should "know" which direction and at what speed it wants to travel. The relationship between agent and goal can range from trivial (standing still) to complex (navigating a maze). Menge is architected in such a way as to easily specify what type of implemented solution to use in any context; one simply references the desired element in the Menge project file. The elements used for plan computation are the `Velocity Component` and `Velocity Modifier`.

`Velocity Component`

The `Velocity Component` element is responsible for computing the agent's preferred velocity; each FSM-state contains one `Velocity Component`. As such, the manner in which a preferred velocity is computed for an agent in one FSM-state can be completely different from that computed for the same agent in a different state. For example, in simulating a train station, a pedestrian would travel to the train platform and then stand and wait for the train. The FSM-state that corresponds to the traversal of the train station would use a `Velocity Component` that can find a path through the complex environment. But the waiting FSM-state can simply produce a preferred velocity sufficient to maintain its position.

Generally the `Velocity Component` implementations primarily define the *direction* of preferred velocity and rely on the agent's own preferred *speed* to specify the magnitude of the preferred velocity vector. However, the interface also allows for a velocity component to arbitrarily deviate from the agent's preferred speed.

Following Curtis et al. [10], preferred velocity is represented by an *arc* of velocities rather than the single vector traditionally used. The arc represents a space of velocities all of which would lead the agent to travel through a space of topologically equivalent paths. The arc is coupled with a function defined over the domain of the arc to distinguish a single "most-preferred" velocity from the space. This preferred velocity arc is the output of the `Velocity Component` and acts as input to the plan adaptation layer. For algorithms which cannot generate such a velocity arc, an arc with a zero-radian span is sufficient. Similarly, if a pedestrian model cannot make use of an arc of velocities, it can operate strictly on the most-preferred velocity from the arc, maintaining the broadest compatibility.

Menge includes many default `Velocity Component` implementations including graph searches on road maps or navigation meshes, straight-to-goal computation, guidance fields, and constant velocities, allowing for the creation of complex scenarios and facilitating the efficient creation of simple scenarios.

`Velocity Modifier`

The `Velocity Modifier` element serves as an interface between the plan computation and plan adaptation modules. The `Velocity Component` is typically an implementation of a global path-planning algorithm concerned with minimizing a property of the path (e.g., length or travel time). The path adaptation uses purely local information to transform the preferred velocity into a feasible velocity. However, this paradigm may be insufficient for modeling behaviors that are dependent on temporal or spatial scopes that lie outside of the global or local path planner. The `Velocity Modifier` element provides a mechanism for introducing additional *layers* of velocity computation. The element receives a preferred velocity as input and transforms the preferred velocity based on its intrinsic algorithm to output a new, modified preferred velocity. A series of `Velocity Modifier` elements can be composed to produce the final preferred velocity used by the plan adaptation stage.

For example, a `Velocity Modifier` element can be used to perform mid-range collision avoidance (e.g., [16,19]); the basic direction of travel to reach the ultimate global goal can be modified according to the presence of other agents beyond the planning horizon of the local collision avoidance. Menge includes modifiers for modeling formations, moving on uneven terrain, and modeling pedestrian density sensitivity (see Section 4.2).

3.8. Plan Adaptation

The preferred velocity computed in the previous section reflects a *static* plan. Dynamic features, such as other agents, may interfere with the execution of that plan. Thus, the preferred velocity needs to be transformed to the next best *feasible* velocity. The definition of "best" and how it is evaluated can be arbitrary. As shown in Section 2.3, there already exist many different models which adhere to this paradigm and, therefore, are compatible with the Menge framework.

`Pedestrian Model`

At its core, a novel `Pedestrian Model` element need only define a single function: the function mapping preferred velocity to feasible velocity. In practice, novel models require their own parameters. As with all other elements, part of the design includes an interface to automatically extend the XML simulator specification to parse and validate required model parameters. Menge's initial release includes several models including two velocity-obstacle-based models and several force-based models. Additional models are forthcoming, including continuum and cellular automata.

3.9. System Elements

The previous elements provide the core behavioral functionality of a Menge simulation. In contrast, the system elements encapsulate the elements which support behavioral computation. This includes the `Spatial Query`, `Elevation`, `Task`, and Event-related elements.

`Spatial Query`

The `Spatial Query` element provides an interface to perform visibility and proximity queries. Implementations of novel spatial query algorithms and data structures can be incorporated in Menge via the plug-in architecture. Menge includes two different implementations: a navigation-mesh centric query class and a kd-tree-centric class. Other spatial queries can be introduced as simulation needs present themselves. For example, it is easy to imagine that in some cases, a simple grid-based solution may be best.

`Elevation`

Menge performs its simulation in two dimensions. Strictly speaking, it can be considered to be a local, two-dimensional manifold in a larger, complex domain. Menge provides the `Elevation` element to provide a mapping from the local 2D planning plane to a complex topology. The `Elevation` element defines the height and the gradient of the domain at an agent's position. Menge's default release includes two `Elevation` implementations: a 2.5D height field and a navigation mesh (which allows for complex, non-planar topologies; see Section 4.1 for an example).

`Task`

The `Task` element is the mechanism by which Menge allows for the insertion of arbitrary, user-defined blocks of work into the simulation pipeline. `Tasks` are evaluated serially in the update stage (as shown in Figure 2). The `Task` can be explicitly instantiated in the simulator specification, or implicitly instantiated in support of another element. For example, algorithms which use a navigation mesh require accurate knowledge of where on the mesh an agent is located. The work to update this information is encoded in a task and executed at the beginning of the pipeline cycle. Even if multiple, independent elements require this work to be done, the shared task guarantees the work is only performed once. Alternatively, a `Task` can be explicitly instantiated by the user in the XML specification.

`Event Triggers, Targets, and Effects`

Menge provides the basis of a complex event system. An event is uniquely defined by three elements: `Event Trigger`, `Event Effect`, and `Event Target`. An event is *triggered* by some specified condition being met. In response its corresponding *effect* is applied to the indicated *target*. The event system has been decomposed in this way to maximize re-use of conceptual blocks. Events complement the BFSM for changing the simulation with respect to time. The BFSM changes the agents behavior based on the agents internal state (e.g., reaching a goal, running out of time, etc.) The event system allows changes to an agent due to factors external to the agent (e.g., a fire happening at a random time).

`Event Triggers` define the conditions for an event to be emitted. The conditions can be defined with respect to any subset of the simulator state. This can include simple timers (such as traffic signals), region population, user actions (in an interactive context), or an `Event Trigger`'s arbitrary internal state. `Event Targets` specify the Menge components upon which the event operates. Events can affect agents, states, or other elements of Menge; one could use an event to dynamically "re-wire" the BFSM. `Event Effects` encode the actual effect of the event when triggered. `Event Effects` can include changing b-State parameters of agents, disabling transitions, terminating the simulation, dynamically blocking pathways, etc.

Finally, Menge's architecture assumes that agents are independent entities. This admits the possibility of extensive, simple parallelization of the algorithms on shared-memory systems. The major stages in the simulation pipeline (such as computing preferred velocity, computing feasible velocity, updating agent state, etc.) are performed in parallel and the pipeline is synchronized at the end of each stage. This gives Menge the potential to be very scalable for many agents on many cores (see Section 4.1 for details).

3.10. Scenario Specification Elements

Menge also provides elements for specifying the initial conditions of the simulation as well as the BFSM. To define the initial conditions of a simulation, each agent's a-state, b-state, and FSM-state are initialized by the `Agent Generator`, `Profile Selector`, and `State Selector` elements, respectively. A group of agents is defined by a triple consisting of an instance of each of those elements. The impassable obstacles in the scene are defined by the `Obstacle Set` element.

`Agent Generator`

The `Agent Generator` is responsible for generating a number of agents and assigning them initial positions and velocities (the agent's a-state). To facilitate the construction of simulation scenarios, Menge provides several implementations ranging from explicit lists of agent positions to abstractions of two-dimensional arrays of agents. Using parametric generators makes experimenting with the simulator simple; one can simply modify the parameters to scale the number of agents in the simulation.

`Profile Selector`

An agent's b-state is defined by an *agent profile*. The agent profile consists of collections of values for b-state parameters. For a given property, the profile can define a value as a global value or drawn from a distribution of values. For example, one could model a crowd of average pedestrians by defining an agent's preferred *speed* with a normal distribution [6] (mean: 1.3 m/s, standard deviation 0.1 m/s). A `Profile Selector` assigns a user-defined profile to each agent. The assignment criteria can be, as with all Menge elements, based on arbitrary user-defined principles. They could be based on initial position in the simulation, count, round-robin assignment, random assignment, etc. Profiles and `Profile Selector` elements permit the user to efficiently create heterogeneous crowds. The populations can easily be varied to facilitate experimentation.

`State Selector`

The `State Selector` is similar to the `Profile Selector`. The `State Selector` assigns an initial state in the BFSM to each agent's FSM-state. As with previous elements, the assignment criteria can be arbitrary. This is particularly important because the BFSM can consist of connected components; not every state may be reachable from an arbitrary start state. These connected components inherently segregate the agents based on behavior. Each connected component defines a unique *category* of agent (e.g, police, pedestrian, etc.) To refer again to the train station, agents can easily be partitioned into initial states which represent having a ticket or not through the use of a `State Selector`. The `State Selector` facilitates the creation of behavioral categories.

`Obstacle Set`

`Obstacle Sets` specify the impassable walls in the simulation. These may be the boundaries of an office building, or hazards which are activated dynamically. `Obstacle Sets` allow for the explicit instantiation of obstacles through vertex lists, or more complex obstacle generation such as capturing obstacles from a navigation mesh or from a geometry file. Novel implementations could create obstacles from any arbitrary construct.

3.11. Extensible XML-based Specification

The XML specification facilitates realizing the design goal of a framework that provides a *low-cost entry*; novel simulation scenarios can be created without writing any C++ code.

---

[6] In fact, Menge uses an approximate normal distribution. Values are limited to the range: $[\mu - 3\sigma, \mu + 3\sigma]$. This prevents unlikely but possibly catastrophic values from being generated.

Menge's XML-based specification language is used to instantiate a simulator session, define the simulation environment, initial conditions, behaviors, and more. However, Menge simultaneously seeks to offer *bespoke functionality* by allowing users to introduce novel elements into the system. These novel elements must also be accessible via the XML specification.

To that end, Menge includes a set of utilities to facilitate the extension of the XML-based specification. Each element definition includes a straightforward interface for communicating to Menge the parameters the element requires and how they should be represented in the XML specification. At run-time, Menge refers to this data to parse the XML and provide the novel element implementation the required data (including detecting if the data is incomplete or incorrectly formatted). In most cases, it is completely unnecessary for a researcher to deal with XML parsing in order to instantiate novel elements from the XML specifications. Alternatively, the element abstraction also provides an alternate, advanced interface which allows for arbitrarily complex XML sub-trees to be parsed by the plug-in. If a complex XML specification is required, the plug-in writer can take the responsibility for parsing it.

4. Application and Evaluation

In this section we examine specific examples which illustrate the Menge's efficacy as a research framework. We focus this discussion on the attached video. We begin with the examples which illustrate the unique benefits of Menge. Then we examine other pedestrian research. We show how various subproblems in pedestrian research can be implemented in Menge. Finally, by implementing these independent works in the Menge framework, we produce a scenario which effectively makes use of otherwise independent research results.

4.1. Illustrative Examples

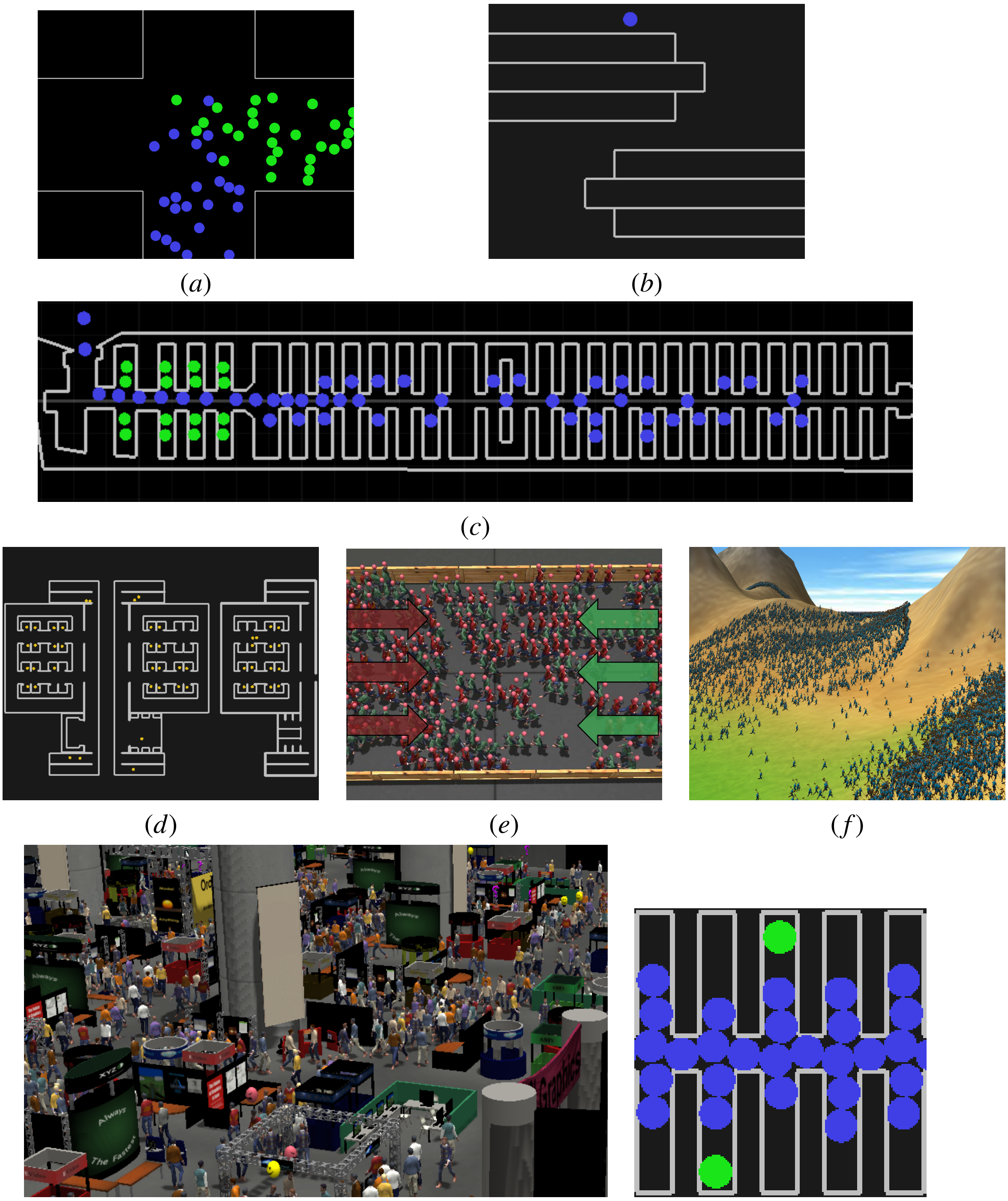
In this section, we draw attention to some of the examples in the accompanying video and show how they illustrate the benefits of Menge. The actual simulated results are available on the Menge website [7].

Cross Flow: The cross flow experiments illustrates a common experiment for pedestrian simulation; two groups of agents move through intersecting, perpendicular hallways (shown in Figure 3(a)). In this example, we vary the `Pedestrian Model` implementation between a velocity-obstacle model [3], a simple social-force model [21], and a predictive social-force model [30]; all other aspects of the simulation are fixed. The differences in behavior due to the `Pedestrian Model` are clear (as illustrated by the sample trajectories shown in Figure 4).

Obstacle Course: The obstacle course experiment compares global planning algorithms. The agents shown in Figure 5 must traverse the scene from top to bottom. This

---

[7] `http://gamma.cs.unc.edu/Menge/`



(*a*) (*b*) (*c*) (*d*) (*e*) (*f*) (*g*) (*h*)

Figure 3 Images from a subset of the various prototype scenarios included with Menge. (*a*) Cross flow highlighting pedestrian model comparisons. (*b*) A benchmark translated from SteerBench XML. (*c*) Airplane loading using random goal selection. (*d*) Agents work at desks and perform other activities in a three-story office building. (*e*) General Adaptation Syndrome algorithm simulation. (*f*) A battle scene showing 32,000 agents moving across complex terrain at interactive simulation rates. (*g*) The trade show scene demonstrating agents moving to and judging exhibits (*h*) Agents (green) waiting for the aisle to clear using a custom transition in an airplane.

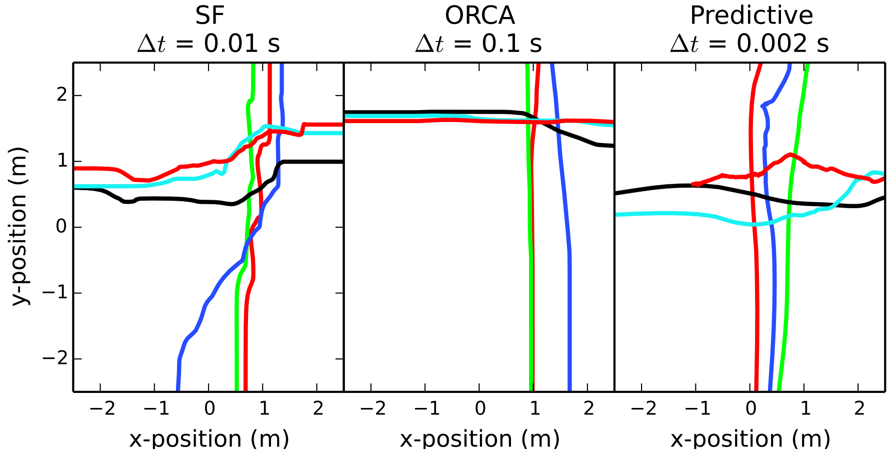


Figure 4 Trajectories plotted for three different pedestrian models in the Cross Flow scenario: a simple social force based model (SF) [21], a velocity-obstacle model (ORCA) [3], and a predictive forces model (Predictive) [30]. With all other simulation elements the same, these trajectories illustrate differences in the model behaviors.

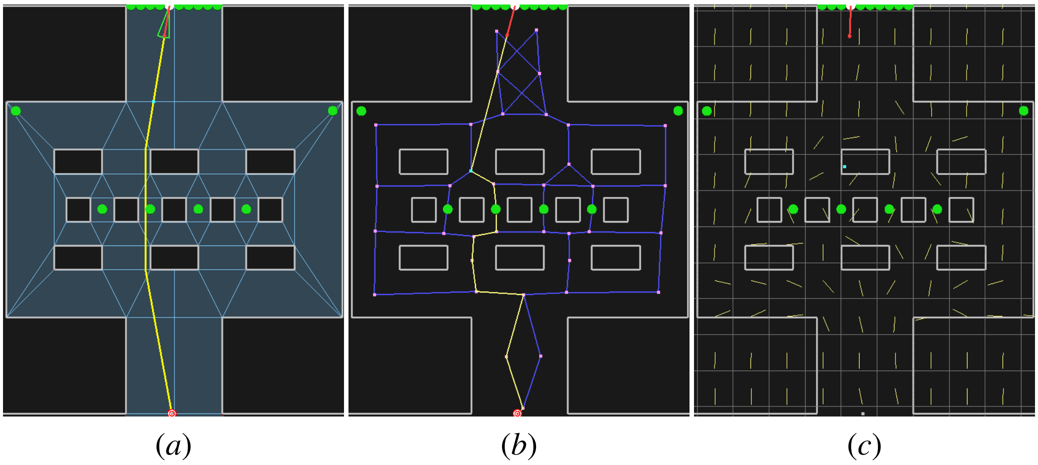


Figure 5 Visualization of three different global navigation methods applied to the Obstacle Course scenario. The green discs are agents; the yellow line represents the path computed for a single agent by each algorithm. In the case of the guidance field, each cell's direction vector is shown in yellow. (*a*) the navigation mesh, (*b*) the roadmap, and (*c*) the guidance field. These navigation structures can be swapped by changing a single line of XML, the `Velocity Component`

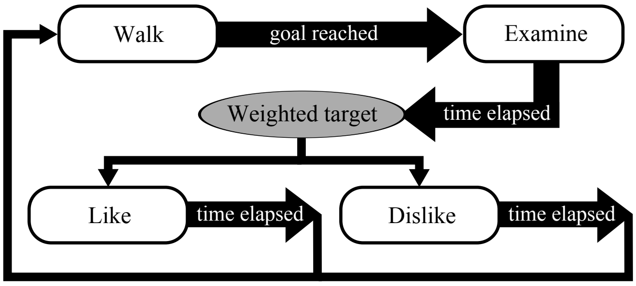


Figure 6 An illustration of the BFSM used in the trade show scenario. The white boxes represent FSM-states, the black arrows represent transition `Conditions`, the grey circle is a transition `Target`. Agents walk to an exhibit. When they reach the exhibit they enter the "Examine" state and stay there for a random amount of time after which they randomly enter the "Like" or "Dislike" state based on weighted probabilities. Finally, after a random amount of time in those states, they select and move to a new exhibit.

time, the `Pedestrian Model` is fixed and the `Velocity Component` changes. We compare a road map, navigation mesh, and guidance field. This experiment, in conjunction with the cross flow experiment, illustrate how Menge facilitates contrasting and comparing algorithms. Menge's formulation of a crowd simulator as a composition of *elements* makes this possible.

SteerBench: The SteerBench scenario illustrates the ease with which scenarios can be defined in Menge's specification language. SteerBench is a set of scenarios designed to evaluate steering algorithms [57]. Each benchmark explores a particular task of crowd navigation and offers a score for an algorithm based on several extensible criteria. The environments, behaviors, and initial conditions of SteerBench are all well expressed in Menge; we use a conversion script to translate from SteerBench XML to Menge's XML specification.

Trade Show: The trade show demo illustrates the principle discussed in Section 3.3 – modeling changes in agent mental state without changes in movement. In this example, we are simulating the behavior of exhibition attendees on the exhibition floor. Agents approach exhibits, examine them briefly, and then decide whether they "like" the exhibit or not. The examination and decision are stationary activities but these activities are encoded as different FSM-states in the BFSM for the agent (shown in Figure 6). In turn, we can use this FSM-state information to visualize their mental state. In the video, we illustrate the examination, approval, and rejection of an exhibit via an icon floating above the agent's head (a question mark, happy face, and angry face, respectively.) This simple visualization hints at what a more sophisticated visualizer could do with the behavior FSM-state information, synthesizing custom behavioral animation that extends beyond mere locomotion.

Battle: The battle scenario demonstrates Menge's scalability and features the `Elevation` and `VelocityModifier` elements. Menge's crowd simulation is not limited to simple planes. Menge agents can move along height fields and, in turn, be affected by those height fields. In this scene, an army of approximately 8,000 agents flee from a pursuing army of $\tilde{2}4$,000 agents. The terrain is defined by a height field. The `Elevation`

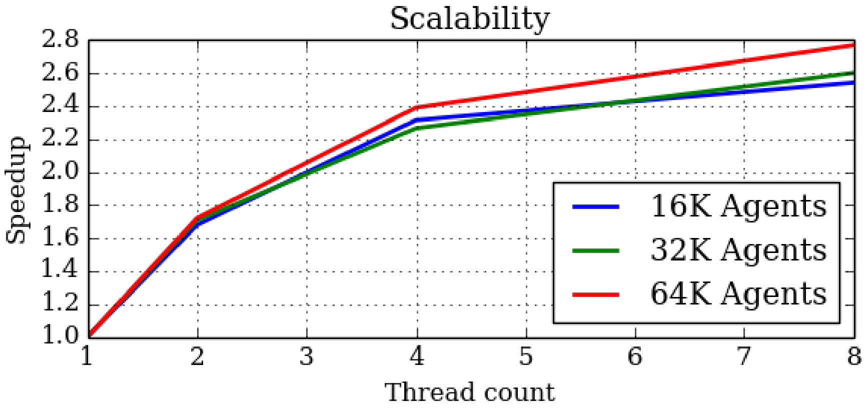


Figure 7 The results of a scalability experiment for Menge. The Battle scene was simulated using 16,000, 32,000, and 64,000 agents, respectively. The simulation used direct to goal navigation with a terrain sensitive `VelocityModifier` and ORCA for local navigation. The average frame computation time was measured based on the number of threads. The speed up over a single thread is shown. The use of a rectangle placement `Agent Generator` makes this experiment simple; we only change the agent count in the XML specification to change the initial conditions.

element places the agents at the appropriate elevation on the terrain. The agents use a simple `Velocity Component` pointing toward a distant goal. However, we have introduced a novel `VelocityModifier` which causes the agents to avoid steep inclines. Together, the agents move towards their goal while adapting to the terrain; agents flow toward valleys and avoid peaks.

This scenario contains the largest population and provides an opportunity to show how Menge scales with population. Figure 7 reports the performance as we varied the population. In its current state, Menge uses primitive locks to maintain safe, concurrent execution. Future versions will include more sophisticated mechanisms and improve Menge's scalability.

Stadium: In this scenario, we reproduce an experiment performed with human subjects: exiting a soccer stadium. This illustrates one way Menge can be used for simulating real-world scenarios. Furthermore, it highlights Menge's ability to perform simulation in complex, three-dimensional scenarios with non-planar topology (illustrated in Figure 8). In this case, the simulation makes use of a navigation mesh structure as part of implementations of a `Velocity Component`, `Elevation`, and `Spatial Query` elements.

Office: The office scenario demonstrates the most complicated BFSM in the set of examples, and shows a practical alternative to simulating complex topologies. Behaviorally, each agent in the scene engages in one of several actions: working at a desk, using the restroom, getting refreshments, leaving the building, and visiting the copy room. To perform the activity, the agent must move to the activity location. Agents can plan across floors to reach the activity location. However, instead of representing the three-story office block literally (i.e., in three dimensions using a complex navigation mesh), we improve the visual clarity of the simulation by laying each floor out on a single plane. This physically disconnects the stairs, but we can account for this by using a teleport `Action`

(*a*) (*b*) (*c*)

Figure 8 Images from the stadium experiment. Pedestrians walk down the aisles and exit through the stadium tunnel. (a) A photo from the original data collected by [7]. (b) A rendered screenshot of the experiment replicated in Menge. (c) The 3D navigation mesh used for the Stadium scenario. The navigation mesh provides elevation information, navigation, and provides a spatial query structure.

to seamlessly move agents traversing the stairs across the discontinuity. We use a road map in the scene and explicitly connect nodes across the disconnected regions. Conceptually, the agents behave the same as if the three floors were stacked on top of each other. This scenario illustrates a combination of goal-choice mechanisms, actions, transitions, and states. It demonstrates Menge's ability to represent populations of agents performing different tasks, with different goals and different strategies all in a single simulation.

4.2. Novel Models in Menge

The previous section illustrates Menge's flexibility in general; abstract scenarios exercise straight-forward algorithms. But Menge can serve as an effective platform for future research as well. To illustrate this, we discuss several bodies of work – some pre-date Menge and we have implemented them in the Menge framework and others have in fact been developed on top of the Menge framework. These examples underscore how flexible the Menge framework is. Finally, we show that by implementing otherwise disparate research in a common framework, we can easily combine them to model never-before seen scenarios.

General Adaptation Syndrome: The work on modeling General Adaptation Syndrome (GAS) by Kim et al. models how humans respond to stress [32]. Essentially, as stress accumulates, people respond by exhibiting more aggression-like behaviors. The authors modeled the accumulation of stress and used work by Guy et al. to model personality changes [18]. Guy et al. performed user studies to correlate agent b-state parameter space with perceptions of personality characteristics. This study was able to suggest a *displacement* vector in b-state parameter space which was the direction of increased aggression. We implemented this in Menge with a custom `Action` which applies the so-called aggression displacement on agents. We assign the `Action` to a stress-inducing FSM-state and include a transition which causes the agent to periodically re-enter the state – shorter periods model a higher rate of stress accumulation, longer periods, a slower accumulation rate. We reproduced one of Kim et al.'s simulation experiments: two groups of agents


(*a*) (*b*) (*c*)

Figure 9 Images from the formation experiments in the video. (*a*) and (*b*) Two formations out of a sequence created by a group of agents moving through space. (*c*) A dense formation of agents navigating around obstacles.

moving in anti-parallel directions in a wide corridor (see Figure 3e. As with the original results, as stress increases, the agent performance (as measured by flow in the corridor) initially improves before eventually breaking down under an excess of stress.

Formations: Although Menge's agents are fundamentally modeled independently, this does not preclude complex, coordinated group behaviors such as those shown in [17, 26, 64]. As a representative sample, we implemented the approach of Gu and Deng to illustrate how easily formations can be introduced into Menge [17]. This approach defines a formation via a canonical collection of *prioritized* points – transform-invariant positions which define the formation. At each time step, the canonical points are mapped to world space and agents are assigned to formation points, in a prioritized manner. See the original authors' work for the exact details [17].

We reproduce this in Menge by introducing two new elements: a `Task` and a `VelocityModifier`. The `Task` is responsible for transforming the canonical formation and mapping agents to formation positions. It executes once per time step, populating a data structure used by the `VelocityModifier`. Agents in a common formation are affected by a common `VelocityModifier`. After each agent computes its own preferred velocity (presumably to the same goal) the `VelocityModifier` modifies it so that it will cause the agent to converge towards its position in the formation. Figure 9 illustrates some of the results using these new elements. In the video, we show one example in which a single group of agents changes formation as it traverses through space and a second example in which a larger formation navigates around obstacles.

Ped-Air: Ped-Air, a simulator described by Best et al. [4], uses Menge to simulate passenger loading, unloading, and evacuation behaviors in aircraft. Simulating passengers on aircraft is challenging for several reasons: passengers can span a broad space of physical and psychological types, they often are pursuing simultaneously contradictory objectives, and they must act in an extremely constrained environment.

Ped-Air exploits Menge's `GoalSelector` element to model passenger seat assignment and to experiment with boarding strategies. The `GoalSelector` defines which seat an agent is heading towards (i.e., its seat assignment). By simply changing the parameters of the `GoalSelector` element, Ped-Air can simulate back-to-front, front-to-back, random, and zone-based seating assignments.

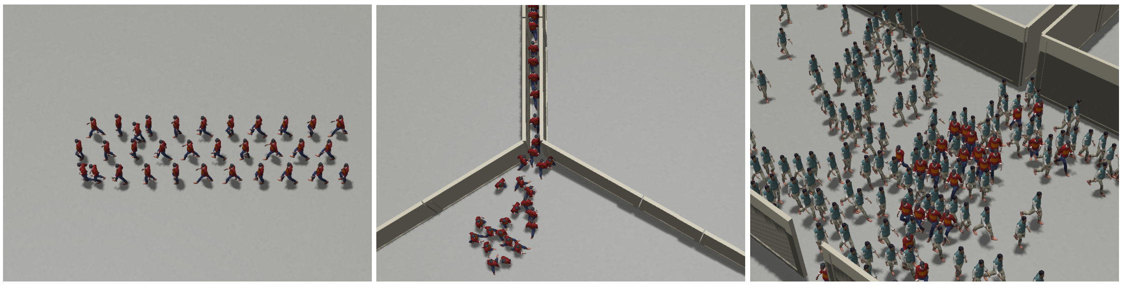
A `GoalSelector` element is also used to model agents stowing luggage in bins. The bin space is discretized into slots with fixed capacity. As each agent boards the plane, it searches for a bin `Goal` near its seat with sufficient capacity for its luggage. The `GoalSelector` easily determines a viable target bin, while constantly accounting for capacity.

When disembarking an airplane, some passengers may remain in their seats until the plane is mostly empty. Ped-Air models this behavior with a custom transition `Condition`. A delaying passenger only transitions from its seated FSM-state to an exiting FSM-state when the aisle forward of its seat is empty of passengers. Furthermore, in some cases, such a passenger requires assistance to disembark. Ped-Air uses custom `Condition` and `Goal` elements to achieve this. When the aisle to a waiting passenger is clear, an agent representing a member of the flight staff moves to the waiting agent. The `Condition` for the waiting agent to begin exiting is that the flight staff agent *reach* it. Then, when the waiting agent begins the exit, the flight staff agent uses a custom `Goal` to accompany the agent; in effect, the exiting agent defines a moving goal for the accompanying agent.

Density-dependent Behaviors: The Fundamental Diagram is a name given to a commonly observed phenomenon in crowd behaviors; as crowds get denser, they get slower [62]. Best et al. propose an algorithm (DenseSense), based on Menge, which successfully reproduces this behavior [5]. The approach works by modifying an agent's preferred velocity based on local density; it operates on the hypothesis that in dense environments, pedestrians are less comfortable moving at high speed. The authors use a relationship between various biomechanical and psychological factors and *preferred velocity* to model this.

Like in the formation work, DenseSense uses a `VelocityModifier` to achieve its goal. The `VelocityModifier` computes the density in an agent's region and then uses it to compute a "comfortable" velocity for the agent to take (see the paper for details). To further optimize this task, it also introduces a new `Task`. At each time step, the custom `Task` computes a density field in the simulation domain. The density field is shared for all agents and the `VelocityModifier` can simply "look up" the density for the agent in question.

Formation Stress: The Formation Stress example underscores Menge's greatest benefit: the simple combination of orthogonal research. In this example, we have combined three separate research results into a single scenario: GAS, formations, and density-dependent behaviors. Because they have been implemented in a common framework, we can author a scenario that makes use of all three. In this scenario, a formation of agents moves towards the entrance of a building. After a predetermined time, an alarm sounds causing the agents to begin accumulating stress. The stress causes the agents to leave their formation and run to the entrance in a chaotic manner, creating a bottleneck at the entrance. After traveling through a short corridor, they enter a large hallway where they must cross through a confused flow of agents, all the while exhibiting the hallmark sensitivity to density seen in real pedestrians (see Figure 10).



(*a*) (*b*) (*c*)

Figure 10 Visualization of the Formation Stress scenario. This brief experiment illustrates the simplicity of combining previously unrelated algorithms to produce a novel simulation. This experiment combines results from stress modeling, formations, and Fundamental Diagram adherence [5, 17,32]. (*a*) The agents travel in a three row formation towards the building. (*b*) After the alarm sounds, the agents run for the entrance, breaking formation. (*c*) When agents reach the corridor after the entryway, they must navigate through a cross-flow, respecting local density constraints on their velocity.

5. Menge's Unique Realization of a Simulation Framework

Menge's architecture realizes the desired framework benefits in the following ways:

- Low-cost entry: Menge is an "out-of-the-box" simulator; users can simulate novel scenarios without writing a line of code. New scenarios, from simple to complex, can be created using only the XML specification language. Menge's behavior FSM allows for complex scenarios, illustrated by the many examples included in the release. The decomposition into elements and the many examples, serve as tutorials to new researchers.

- Focused development: Researchers can make use of Menge's element-based design to focus their efforts on specific aspects of crowd simulation, relying on the remaining implementations to provide a robust context to test their models.

- Efficient dissemination: Novel crowd models can be released as Menge plug-ins, such as the formations and density-sensitivity functionality discussed above. Other Menge users can include the plug-ins in their own builds and immediately make use advances in the state of the art.[8] Authors can host their plug-ins themselves, or new functionality can be rolled back into the Menge release, facilitating sharing.

- Meaningful comparison: Menge's modular approach particularly facilitates comparisons. For a given simulation scenario, two competing algorithms can be compared, simply by changing the reference in the XML configuration. As shown in Section 4.1 with respect to pedestrian models and global navigation. Alternatively, in many cases, both algorithms could be used in a single simulation scenario.

---

[8] Although binary distributions are strictly possible, due to compatibility issues in C++ runtime libraries, release of the source code is preferred.

- Bespoke functionality: The elements in Menge have interfaces designed with the intent to be as simple as possible – the simpler the interface, the simpler the integration. Furthermore, Menge includes utilities to ease the task of extending the XML specification to include new element implementations.

- Flexible specification: Without compiling Menge, a user can make significant changes in what is simulated, how it is simulated, and how the results are stored. Menge's XML simulation scenario specification includes more than just the utilities for testing simulation models. It includes mechanisms for efficiently defining initial conditions, specifying simulation parameters such as time step and duration, caching simulation results, and controlling the visualization.

We have shown how Menge can serve as an effective framework for the crowd simulation community. However, Menge is not the only crowd simulation software available. Even putting aside the many commercial simulation packages, there are still many open-source software package targeted towards academia. We have examined a number of such tools and illustrate that while they have many strengths – some quite unique and valuable – none of them can serve as a framework to the same degree as Menge. Although we have made a concerted effort to provide the most complete survey possible, we limit this discussion to those applications targeted towards the research community, are currently available, and are "extensible", in some sense.

All of these applications provide a *reduced entry cost* to varying degrees; simply having access to pre-existing software provides a new researcher an advantage. The utility of this starting point, however, depends on what the researcher intends to do with the application, and how compatible the application is with that intent. This ability to introduce new behavior into the simulation application is what we mean by "extensible". Two ways for an application to be extensible is to be open-source or to allow plug-ins (or both).

Open-source applications are extensible because users can modify the code to suit their purposes. However, the architecture of the application will necessarily render some modifications easier than others. If the original authors intend a particular feature to be modified, then the code will facilitate this action. However, the converse is likely true – aspects of the system the original authors expected to go untouched are likely to be tightly coupled and replacing them will be more challenging. This type of extensibility, while empowering isolated research groups, is less effective for dissemination of results. Two research groups may go about modifying the original system in two different ways to incorporate their novel models. Merging such results into a common framework may prove to be problematic.

A plug-in-based application has strongly formalized extensibility. It has the same pitfalls as "open-source extensibility"; only those aspects the original authors anticipated as needing to be modified would be included in the plug-in interface. However, the plug-in architecture greatly facilitates result dissemination. The plug-in interface defines an isolating layer – on one side lies the application, on the other side lies the novel contributions. The plug-in can be released either as code or as binaries. This is one reason why Menge has a plug-in architecture and dozens of components are exposed in this interface.

We have found five open-source crowd simulation applications targeted towards the research community. Below we give a brief overview of each application and our best judgment in how well it provides the desired benefits of a common simulation framework.

PedSim is the simplest of the available applications [15] discussed here. At its core, PedSim is an small C++ library consisting of a social-force-based steering model and definitions of paths and obstacles. It has no global planning ability, no high-level behaviors, no events, and no goals. It has a simple application which can parse a lightweight specification file to initialize the simulation; agents are defined in place along with their paths. It's social force model includes more features than a "basic" social force model, such as a "look ahead" and a "follow" force. PedSim does not facilitate *focused development* because it is so simple; one cannot rely on components that do not exist. That same simplicity limits *efficient dissemination* and *meaningful comparisons* because so much would have to be invented to mature it into a fully-fledged crowd simulator that it is unlikely that any two groups would do this work in a compatible way. However, it is worth noting that the limited functionality is fully accessible through its scenario configuration.

OpenSteer is a C++ application for exploring steering behaviors [51]. OpenSteer uses a plug-in architecture to introduce various scenarios such as, capture the flag, multiple pursuit, boids [50], waypoint following, soccer, and, curiously, pedestrians. The application is not specifically designed to simulate human pedestrians and is reflected in the architecture – a pedestrian derives from a "simple vehicle" class. Like PedSim, the framework omits a number of capabilities: no behaviors, global planning, events, etc. It is completely focused on evaluating steering behaviors for "vehicles". The compile-time plug-ins encode a particular steering behavior and two behaviors cannot co-exist in a single simulation. As with PedSim, its simplicity precludes its ability to serve as a framework for general pedestrian simulation rendering it unusable from a *focused development*, *efficient dissemination*, or *meaningful comparison* perspective. Furthermore, it has no external specification mechanism; simulation scenarios must be hard-coded into the plug-in.

ADAPT's primary focus is on the final stage of simulation: pedestrian visualization and motion synthesis [28]. It uses Rekast, an open-source navigation mesh and steering algorithm as the underlying planner and pedestrian model [40]. ADAPT's unique contribution is in its behavior tree for controlling articulated pedestrian visualizations through, what the authors term, *coordinators* and *choreographers*. The final, articulated pedestrians can exhibit fine-detailed animations such as sitting, reaching, upper-body gestures, etc. ADAPT's framework relies heavily on the Unity game engine to handle the visualization and motion synthesis as controlled by their behavior tree. As such, its core functionality is authored in C#. *Focused development* is problematic; ADAPT's focus is on the visualized motion synthesis. There is excellent infrastructure for introducing new choreographers, but no allowance for other aspects of crowd simulation, e.g., alternative steering or planning algorithms. As long as researchers are focused on motion synthesis, ADAPT can be effective in *disseminating* novel work. New classes which fit into the behavior framework can be distributed directly and other researchers can compile it into their own version of ADAPT. However, modifying other aspects of the simulator will require custom modifications which may not be compatible from one research group to another. ADAPT is not well suited to model *comparisons*. Scenarios are defined in code, so to compare two different models, two different compilation paths must be maintained to build two different binaries. Generally, when an application has been designed with tightly integrated components, replacing those components becomes logistically awkward. And any endeavor trying to compare and contrast those components will exhibit unwieldiness. Bespoke functionality *can* be introduced into the existing framework but, again, except for motion synthesis, there are no designed vectors for introducing other novel functionality. Finally, Adapt has no apparent external simulation configuration; particular simulation scenarios must be defined in code to be included in compile time.

JuPedSim [33] is the successor to the former OpenPedSim [12] and shares many common features. JuPedSim is a C++ pedestrian simulator apparently targeted towards evacuation scenarios. JuPedSim has no high-level behavior module; simulated scenarios consist of the agents starting from an initial condition and moving toward a final goal. Navigation is handled through 3D navigation mesh-like algorithm and the steering is handled by the generalized centrifugal-force model [8]. Consistent with the apparent problem domain of evacuation scenarios, the simulation supports dynamic environments (e.g., doors blocking and unblocking) and the concept of pedestrian *knowledge* and how it moves through the crowd (useful for simulating evacuation in smoke-filled environments). JuPedSim includes tools for analyzing the results of the simulation (e.g., density analysis and computing the so-called “fundamental diagram”.) The various components of the simulator are tightly coupled which limits the ability to perform *focused development*. For example, the knowledge model is implemented directly into the pedestrian model. If a new researcher wanted to investigate a different pedestrian model with the same knowledge, or a different model of how knowledge is transmitted, it would require extensive coding to pair new components in place of the current pairing. When it comes to *efficient dissemination*, *meaningful comparisons*, and *bespoke functionality*, JuPedSim exhibits many of the same issues as ADAPT. Without a plug-in architecture, there is no designed interface for introducing novel models, so new models would require a custom harness. Sharing novel models predicated on different harnesses is only slightly better than sharing code across independent frameworks. However, JuPedSim has an extensive XML-based specification language for fully exercising its functionality, giving it *flexible specification*.

SteerSuite [56] and Menge have the most similar architectural designs. The C++ application includes the ability to use run-time plugins to modify the behavior of the simulator. These plug-ins can be used to change the pedestrian model, global navigation algorithm, spatial query mechanism, and change how the simulation is visualized. SteerSuite includes an XML-based specification for designing new simulation scenario and includes a suite of simple scenarios for evaluation of novel scenarios. Like ADAPT, SteerSuite has included the Rekast [40] code as one of the global planning algorithms. The framework also includes instrumentation for performance profiling. However, SteerSuite has no high-level behaviors. As with the previous applications, simulation scenarios consist of agents in initial positions moving toward a fixed goal. The global navigation algorithm is global – all agents in the simulation must use the same mechanism to move toward their goal. Finally, the XML specification is not extensible; referencing novel components in the specification would require modifications to the core application. SteerSuite provides a strong base for *focused development*; specific global planning or steering algorithms can be inserted into the system, relying on the other, pre-existing components. The same plug-in architecture enables *efficient dissemination* and *meaningful comparisons* because novel models can be distributed and built independent of the SteerSuite source code. And a single simulation scenario can be run multiple times with different pedestrian models by specifying the plug-in to apply. SteerSuite allows for *bespoke functionality* in steering and global navigation algorithms, but research into high-level behaviors would require completely new code to interface with the current architecture. Finally, the XML scenario specification gives limited access to the simulation constructs, indicatig limited *specification flexibility*.

All of these applications are available and will serve their specific purpose. A researcher, looking to enter the domain of pedestrian simulation, could select one and benefit from using it as a starting point. They each have a unique strength borne of an apparent targeted, intent. JuPedSim has modeled spatial awareness and knowledge propagation. ADAPT has high fidelity crowd visualization and state-based motion synthesis. SteerSuite has a suite of scenarios and comes ready with multiple pedestrian models implemented. OpenSteer explores a number of behavioral scenarios (e.g., capture the flag, etc.) The fact that these features are spread out across multiple applications is precisely the functional fracturing that occurs in the absence of a common framework. This is regrettable because these features are not mutually exclusive; the could happily co-exist in a single framework.

Menge's initial implementation includes a subset of these features. However, the underlying architecture and design of Menge makes it possible to incorporate *all* of these features in future releases. Furthermore, Menge's extensibility is different from the other applications which have implemented plug-in architecture. Where other systems use plug-in framework to introduce *replacements* for the built-in functionality, Menge's architecture uses the plug-ins to *extend* its functionality; multiple independent element implementations co-exist in the system and can be used in a common simulation scenario (as shown in Section 4.1).

6. Final Remarks

6.1. Conclusion

We have presented the design of a novel, modular framework for the simulation of crowd movement. Through the combination of various modular constructs, called *elements*, novel crowd simulators can be dynamically constructed to simulate a wide range of scenarios and behaviors. Furthermore, because of its plug-in architecture, particular implementations of Menge elements can be released as code or binary objects, enabling users of the framework to share their own advances and benefit from the contributions of others. We have discussed the validity of Menge's paradigm in the context of representative samples from crowd simulation literature and shown, through specific examples, the strengths and properties of this framework.

Menge provides a platform for crowd research that facilitates straightforward combinations of algorithmic techniques that were previously infeasible. Rather than producing algorithms which target a particular subproblem in crowd simulation without consideration of how those algorithms fit into a larger context, Menge encourages researchers to produce algorithms which are inter-operable and provides algorithmic implementations which are themselves inter-operable. Researchers have the opportunity to build on common work in a way not previously available to them. Simulators can be constructed in Menge that take advantage of a number of models which would not otherwise be compatible without such a common core framework upon which to build.

Menge is open-source, cross-platform, and publicly available [9]. Ultimately, we hope that the adoption of a framework such as Menge, would foster tighter integration among the crowd simulation community. New researchers would enter the domain able to exploit the current state of the art and directly apply their efforts to novel algorithms. Published work could be closely supported by the releases of supporting code or binaries for the community's benefit and future comparisons.

6.2. Future Work

Menge is a work in progress and has definite limitations. First, it currently only allows one mechanism for generating high level behaviors – the BFSM. Behavior trees are a common structure in game AI [49]. Both approaches essentially encode agent behavior in graph nodes, but they largely differ in how the network of nodes is traversed. Currently, this traversal is not an exposed part of the Menge interface, rendering behavior trees unusable.

Second, planning and personality are tightly coupled in the BFSM. A single FSM-state specifies both *what* the agent seeks to accomplish and *how* (i.e., its personality and mood). While this does not actually limit Menge's ability to model complex scenarios, it can make the task more difficult, requiring redundancies in the specification where two FSM-states share the same objective but possess different behavioral profiles.

Menge has implicitly excluded the subproblem of motion synthesis, but Menge's architecture does not prevent a `Pedestrian Model` implementation from considering biomechanical factors in adapting preferred velocity. Menge would certainly benefit from the inclusion of a system for synthesizing motion in a modular manner similar to the other elements.

Additionally, Menge is an agent-based crowd simulation framework. Some recent work, including [55] and [24], uses motion-patches to create populated scenes of pedestrians. These methods create agents as needed to fill motion scripts and do not contain agents exploring shared spaces and planning/interacting as they accomplish disparate goals. Although a `Pedestrian Model` and `Velocity Component` could be implemented that compute paths for agents with respect to a predefined set of motion-patches, this would be a substantial undertaking.

Menge's implementation is in its infancy. As such, there are some short-term implementation issues which limit its utility. As previously noted, it uses a primitive paral-

---

[9] `http://gamma.cs.unc.com/Menge/`

lelism mechanism which causes its scalability to suffer. In addition, Menge simulations use a fixed population; there is no mechanism in place for removing or introducing agents during the course of the simulation. Menge's core element implementations have been written with this eventual functionality in mind, so that it can be introduced in the future without losing backwards compatibility with prior implementations.

In the future, Menge will seek to address these limitations and others as the community explores spaces as yet unconsidered. We invite others to explore the Menge framework and produce novel implementations of the many elements. We hope that Menge's future growth will be fueled by groups around the world expanding its feature set according to their varied needs. We invite those eager to contribute; contact information can be found on Menge's website.[10]

References

[1] Bandini, S., Federici, M., Manzoni, S., Vizzari, G.: Towards a methodology for situated cellular agent based crowd simulations. Engineering societies in the agents world VI pp. 203–220 (2006)

[2] van Basten, B.J.H., Stuvel, S.A., Egges, A.: A hybrid interpolation scheme for footprint-driven walking synthesis. Graphics Interface pp. 9–16 (2011)

[3] van den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. In: Inter. Symp. on Robotics Research, pp. 3–19 (2011)

[4] Best, A., Curtis, S., Kasik, D., Senesac, C., Sikora, T., Manocha, D.: Ped-air: a simulator for loading, unloading, and evacuating aircraft. In: 7th International Conference on Pedestrian and Evacuation Dynamics. Delft, Netherlands (2014)

[5] Best, A., Narang, S., Curtis, S., Manocha, D.: Densesense: Interactive crowd simulation using density-dependent filters. In: Symposium on Computer Animation (2014)

[6] Bruderlin, A., Calvert, T.W.: Goal-directed, dynamic animation of human walking. In: Proc. of SIGGRAPH '89, pp. 233–242 (1989)

[7] Burghardt, S., Klingsch, W., Seyfried, A.: Analysis of flow-influencing factors in mouths of grandstands. In: Pedestrian and Evacuation Dynamics (2012)

[8] Chraibi, M., Seyfried, A., Schadschneider, A.: Generalized centrifugal-force model for pedestrian dynamics. Phys. Rev. E **82**(4), 046,111 (2010)

[9] Cohen-Or, D., Chrysanthou, Y., Silva, C., Durand, F.: A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphics **9**(3), 412–431 (2003). `doi:10.1109/TVCG.2003.1207447`

---

[10] `http://gamma.cs.unc.edu/Menge`

[10] Curtis, S., Snape, J., Manocha, D.: Way portals: Efficient multi-agent navigation with line-segment goals. Proc. of Symposium on Interactive 3D Graphics and Games pp. 15–22 (2012)

[11] Duives, D.C., Daamen, W., Hoogendoorn, S.P.: State-of-the-art crowd motion simulation models. Transportation Research Part C: Emerging Technologies **37**(0), 193–209 (2013). `doi:10.1016/j.trc.2013.02.005`

[12] Forschungszentrum, J.: Open ped sim. `http://sourceforge.net/projects/openpedsim/` (2014)

[13] Funge, J., Tu, X., Terzopoulos, D.: Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In: Proc. of SIGGRAPH, pp. 29–38 (1999)

[14] Geraerts, R., Kamphuis, A., Karamouzas, I., Overmars, M.: Using the corridor map method for path planning for a large number of characters. In: Motion in Games, pp. 11–22. Springer, Heidelberg (2008)

[15] Gloor, C.: Pedsim: A microscopic pedestrian crowd simulation system. `http://pedsim.silmaril.org/` (2003)

[16] Golas, A., Narain, R., Lin, M.: Hybrid long-range collision avoidance for crowd simulation. Proc. of Symposium on Interactive 3D Graphics and Games (I3D) pp. 29–36 (2013)

[17] Gu, Q., Deng, Z.: Generating freestyle group formations in agent-based crowd simulations. IEEE Computer Graphics and Applications **33**(1), 20–31 (2013). URL `http://doi.ieeecomputersociety.org/10.1109/MCG.2011.87`

[18] Guy, S.J., Kim, S., Lin, M.C., Manocha, D.: Simulating heterogeneous crowd behaviors using personality trait theory. In: SCA ’11, pp. 43–52 (2011)

[19] He, L., van den Berg, J.: Meso-scale planning for multi-agent navigation. In: Proc. IEEE Int. Conf. on Robotics and Automation - ICRA (2013)

[20] Heck, R., Gleicher, M.: Parametric motion graphs. In: Proceedings of Symposium on Interactive 3D Graphics and Games, pp. 129–136 (2007)

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

[22] Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Physical Review E **51**(5), 4282–4286 (1995)

[23] Hutton, A.: London bridge station, the role of ped modelling: Pedestrian modelling and design development. In: 6th International Conference on Pedestrian and Evacuation Dynamics. Zurich, Switzerland (2012)

[24] Hyun, K., Kim, M., Hwang, Y., Lee, J.: Tiling motion patches. IEEE Trans. Vis. Comput. Graph **19**(11), 1923–1934 (2013). URL `http://doi.ieeecomputersociety.org/10.1109/TVCG.2013.80`

[25] Jaillet, L., Simeon, T.: A PRM-based motion planning for dynamically changing environments. In: Proc. IEEE RSJ Int. Conf. Intell. Robot. Syst., vol. 2, pp. 1606–1611 (2004)

[26] Ju, E., Choi, M.G., Park, M., Lee, J., Lee, K.H., Takahashi, S.: Morphable crowds. ACM Trans. Graph **29**(6), 140 (2010). URL `http://dx.doi.org/10.1145/1882261.1866162`

[27] Kallman, M., Mataric, M.: Motion planning using dynamic roadmaps. In: Proc. IEEE Int. Conf. Robot. Autom., vol. 5, pp. 4399–4404 (2004)

[28] Kapadia, M., Marshak, N., Shoulson, A., Badler, N.I.: ADAPT: The agent development and prototyping testbed. IEEE Transactions on Visualization and Computer Graphics **20**(7), 1035–1047 (2014). `doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2013.251`

[29] Kapadia, M., Singh, S., Hewlett, W., Faloutsos, P.: Egocentric affordance fields in pedestrian steering. In: Proceedings of the Symposium on Interactive 3D Graphics and Games, pp. 215–223 (2009)

[30] Karamouzas, I., Heil, P., van Beek, P., Overmars, M.H.: A predictive collision avoidance model for pedestrian simulation. In: Motion in Games, *Lecture Notes in Computer Science*, vol. 5884, pp. 41–52. Springer (2009)

[31] Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. Int. J. Robot. Res. **5**, 90–98 (1986)

[32] Kim, S., Guy, S.J., Manocha, D., Lin, M.C.: Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. ACM Symposium on Interactive 3D Graphics and Games pp. 55–62 (2012)

[33] Klupfel, H.: `http://sourceforge.net/projects/openpedsim/` (2014)

[34] Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. ACM Trans. Graph. **21**(3), 473–482 (2002)

[35] Kretz, T.: Pedestrian traffic: on the quickest path. Journal of Statistical Mechanics: Theory and Experiment **2009**(03), P03,012 (2009). `doi:10.1088/1742-5468/2009/03/P03012`

[36] Lamarche, F., Donikian, S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. Computer Graphics Forum **23**(3), 509–518 (2004). `doi:10.1111/j.1467-8659.2004.00782.x`. URL `http://dx.doi.org/10.1111/j.1467-8659.2004.00782.x`

[37] Latombe, J.C.: Robot Motion Planning. Springer, Heidelberg (1991)

[38] Lee, K.H., Choi, M.G., Hong, Q., Lee, J.: Group behavior from video: a data-driven approach to crowd simulation. In: Symposium on Computer Animation, pp. 109–118 (2007)

[39] Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of english: The penn treebank. Comput. Linguist. **19**(2), 313–330 (1993). URL `http://dl.acm.org/citation.cfm?id=972470.972475`

[40] Mononen, M.: Recast: navigation-mesh construction toolset for games. `http://code.google.com/p/recastnavigation/` (2009)

[41] Multon, F., France, L., Cani-Gascuel, M.P., Debunne, G.: Computer animation of human walking: a survey, vol. 10, pp. 39–54 (1999)

[42] Narain, R., Golas, A., Curtis, S., Lin, M.C.: Aggregate dynamics for dense crowd simulation. ACM Trans. Graph. **28**, 122:1–122:8 (2009)

[43] Newell, A.: Unified theories of cognition. Harvard University Press, Cambridge, MA, USA (1990)

[44] Oliva, R., Pelechano, N.: Automatic generation of suboptimal navmeshes. In: J.M. Allbeck, P. Faloutsos (eds.) MIG, *Lecture Notes in Computer Science*, vol. 7060, pp. 328–339. Springer (2011). URL `http://dx.doi.org/10.1007/978-3-642-25090-3`

[45] Ondřej, J., Pettré, J., Olivier, A.H., Donikian, S.: A synthetic-vision based steering approach for crowd simulation. In: Proc. SIGGRAPH, pp. 123:1–123:9 (2010)

[46] Paris, S., Donikian, S.: Activity-driven populace: A cognitive approach to crowd simulation. Computer Graphics and Applications, IEEE **29**(4), 34–43 (2009)

[47] Pelechano, N., Allbeck, J., Badler, N.: Controlling individual agents in high-density crowd simulation. In: Symposium on Computer Animation, pp. 99–108 (2007)

[48] Pettré, J., Ondřej, J., Olivier, A.H., Cretual, A., Donikian, S.: Experiment-based modeling, simulation and validation of interactions between virtual walkers. In: Symposium on Computer Animation, pp. 189–198 (2009)

[49] Rabin, S.: AI Game Programming Wisdom 4 (AI Game Programming Wisdom (W/CD)), 1 edn. Charles River Media (2008)

[50] Reynolds, C.: Flocks, herds and schools: A distributed behavioral model. In: Proc. of SIGGRAPH (1987)

[51] Reynolds, C.W.: Steering behaviors for autonomous characters. Game Developers Conference (1999)

[52] Samet, H.: Foundations of MultiDimensional and Metric Data Structures. Morgan Kaufmann (2006)

[53] Schadschneider, A.: Cellular automaton approach to pedestrian dynamics - theory. Pedestrian and Evacuation Dynamics pp. 75–86 (2002)

[54] Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: Symposium on Computer Animation, pp. 19–28 (2005)

[55] Shum, H.P.H., Komura, T., Shiraishi, M., Yamazaki, S.: Interaction patches for multi-character animation. ACM Trans. Graph **27**(5), 114 (2008). URL `http://doi.acm.org/10.1145/1457515.1409067`

[56] Singh, S., Kapadia, M., Faloutsos, P., Reinman, G.: An open framework for developing, evaluating, and sharing steering algorithms. In: Proceedings of the 2nd International Workshop on Motion in Games, pp. 158–169 (2009)

[57] Singh, S., Kapadia, M., Faloutsos, P., Reinman, G.: Steerbench: a benchmark suite for evaluating steering behaviors. Computer Animation and Virtual Worlds **20**(5-6), 533–548 (2009). `doi:10.1002/cav.277`

[58] Snook, G.: Simplified 3D movement and pathfinding using navigation meshes. In: Game Programming Gems, chap. 3, pp. 288–304. Charles River, Hingham, Mass. (2000)

[59] Sun, H.C., Metaxas, D.N.: Automating gait generation. In: Proc. of ACM SIGGRAPH, pp. 261–270 (2001)

[60] Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. In: Proc. of ACM SIGGRAPH, pp. 1160–1168 (2006)

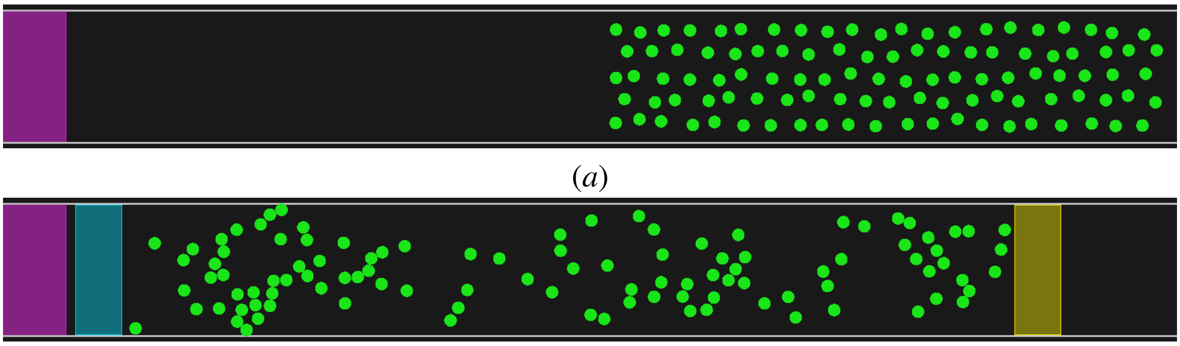
[61] Ulicny, B., Thalmann, D.: Towards interactive real-time crowd behavior simulation. Computer Graphics Forum **21**(4), 767–775 (2002)

[62] Weidmann, U.: Transporttechnik der fussgaenger. Tech. Rep. 90 (1993)

[63] Yang, Y., Brock, O.: Elastic roadmaps: globally task-consistent motion for autonomous mobile manipulation. In: Proc. Robot. Sci. Syst., pp. 279–286 (2007)

[64] Zhang, P., Liu, H., Ding, Y.h.: Crowd simulation based on constrained and controlled group formation. The Visual Computer pp. 1–14 (2013). `doi:10.1007/s00371-013-0900-7`. URL `http://dx.doi.org/10.1007/s00371-013-0900-7`

[65] Zheng, X., Zhong, T., Liu, M.: Modeling crowd evacuation of a building based on seven methodological approaches. Building and Environment **44**(3), 437–445 (2009). `doi:10.1016/j.buildenv.2008.04.002`



(*a*)

(*b*)

Figure 11 The scenario described in the appendix. Agents move from right to left down a corridor. The *effect* of periodic boundaries is realized with a teleport `Action`. As agents move toward the left-hand purple goal region, they enter the cyan box immediately preceding it. Upon entering this region, the agents are teleported back to the yellow region on the right. This motion allows agents to walk down the corridor indefinitely – well approximating periodic boundaries.

A. Menge Simulation Specification Example

Here we give an example of Menge's simulation specification language. We present and discuss the complete description of a simple scenario: uni-directional flow down a corridor with periodic boundaries. Figure 11 shows the initial condition and some later point in the simulation. See below for a detailed description of the figures.

A.1. Scene Specification

Listing 1 provides the complete scene specification. It is responsible for defining the agent population and initial state.

Line 1 The root element of the specification XML.

Line 2 The declaration of the `SpatialQuery` type – in this case, a kd-tree.

Lines 4-6 The specification of the global `Pedestrian Model` parameters, including those shared by all pedestrian models (`Common`) and those particular to the simple social force model (`Helbing`) and the predictive social force model (`Karamouzas`)[11].

Lines 8-15 The definition of an "agent profile", defining the space of values of agent b-state parameters. The profile is named, `group1`, for reference purposes.

Lines 9-14 The per-agent `Pedestrian Model` parameters, including the shared parameters (`Common`), and for three particular implementations (`Helbing`, `Karamouzas`, `ORCA`).

---

[11] The ORCA model does not have any global parameters.

Line 10 The property allows for definitions of b-state parameters using a numerical distribution. In this case, the preferred speed is defined as a normal distribution with a mean value and standard deviation of 1.3 m/s and 0.15 m/s, respectively.

Lines 17-21 The instantiation of a group of agents. The number and position of each agent is defined by the `Generator`, assigned b-state parameter values by its `ProfileSelector`, and assigned an initial FSM-state by its `StateSelector`.

Line 18 The `ProfileSelector` uses a `const` type. Which means that all agents will be assigned the `group1` agent profile. In contrast, distribution-style `ProfileSelector` could assign a profile from a set of specified profiles.

Line 19 The `StateSelector`, like the `ProfileSelector`, is of `const` type and assigns all agents to the same initial FSM-state.

Line 20 The `AgentGenerator` instantiates a hexagonal lattice of agent positions. The arguments specify the geometry of the lattice, average density, and the approximate count of agents. In addition, it provides a displacement distribution to perturb the initial positions from the perfect lattice positions. The noisy lattice can be seen in Figure 11(*a*).

Lines 23-30 These define the obstacles in the environment. In this case, the type of the `ObstacleSet` is `explicit`; each obstacle is explicitly defined in the specification file (in contrast to being read from an external file).

Lines 24-29 The definition of a single obstacle. The obstacle is a closed polygon, defined by a two-dimensional vertex list. The order of vertices defines the "inside" and "outside" of the obstacle.

A.2. Behavior Specification

The behavior specification includes the explicit instantiation of a particular BFSM, as well as supporting data structures. Listing 2 contains the full BFSM specification for the example scenario. The key feature to this BFSM is the teleport `Action` element on line 13. This is what creates the effect of periodic boundary conditions.

Line 1 The root element of the behavior specification XML.

Lines 2-4 The definition of a set of goals. A behavior specification can contain any number of such sets. Each goal set contains one or more `Goals`. Each goal set must possess a unique, numerical id for referencing by other entities.

Line 3 The single `Goal` defined in this scenario. In this case, the `Goal` is an `AABB` (axis-aligned bounding box). Agents will always move to the closest point in the goal region. The box is shown as the purpose region in Figure 11.

Lines 6-9 The definition of the "walking" FSM-state. Uniquely identified by the name `Walk`. The state also indicates that it is a non-final state – the simulation will not end if there are agents in this FSM-state.

Line 7 The `GoalSelector` for this FSM-state. When agents enter the state, they are assigned a `Goal`. In this case, every agent explicitly is assigned a specific `Goal` from a specific goal set (`Goal` 0 from goal set 0).

Line 8 The `VelocityComponent` which causes agents to move directly toward their `Goal`. In this simple scenario, no more sophisticated mechanism is necessary beyond simply walking straight to the goal.

Lines 10-14 The definition of the "goal reached" FSM-state. This state serves a single purpose, to discontinuously move (teleport) agents to a target region. Its various components will reflect this purpose.

Line 11 This FSM-state's `GoalSelector` is of type `identity`. This means that each agent's `Goal` is the point at which the agent is when it enters the state. This is useful for causing agents to hold position.

Line 12 This FSM-state's `VelocityComponent` is the `zero` type. Every agent in this state will have the zero preferred velocity.

Line 13 The `teleport Action` is assigned to this FSM-state. When agents enter this FSM-state, the action is applied and the agents are moved to a random point inside the box implied by the `min_x`, `max_x`, `min_y`, and `max_y` parameters (shown in yellow in Figure 11(*b*).

Lines 16-18 The definition of the transition from the `Walk` to `GoalReached` FSM-states. This makes use of the implied transition `Target` element.

Line 17 The transition `Condition` which causes an agent to move FSM-states. This transition is taken when the agent enters an AABB. The region is shown as the cyan box on the left in Figure 11(*b*). Because of this transition, no agent will ever actually reach its goal, but will, instead, be teleported back to the yellow region.

Lines 19-21 The definition of the transition from the `GoalReached` back to `Walk` FSM-states.

Line 17 This transition `Condition` is an `auto` condition; it is the tautology. It implies that any agent entering the `GoalReached` FSM-state will automatically be transitioned to the `Walk` FSM-state. This type of automatic transitions allows FSM-states to be introduced which have a one-time effect. This transition is also the reason why the `GoalSelector` and `VelocityComponent` in the `GoalReached` state are immaterial; they will never really be used.

A.3. Additional Documentation

The examples provided with Menge illustrate the various methods of creating and running scenes. Complete documentation of the Menge codebase is available at the project website, `http://gamma.cs.unc.edu/Menge/`.

- An installation guide and Getting Started is available at `http://gamma.cs.unc.edu/Menge/learn/gettingStarted.html`
- Documentation on the Namespaces in Menge can be found at `http://gamma.cs.unc.edu/Menge/docs/code/menge/html/namespaces.html`.
- A complete class reference can be found at `http://gamma.cs.unc.edu/Menge/docs/code/menge/html/classes.html`.
- Documentation on the plugins included with Menge can be found at `http://gamma.cs.unc.edu/Menge/docs/code/PedPlugins/html/index.html`

A.4. Specification XML Files

Listing 1 Scene specification for a periodic hallway

```
<Experiment version=”2.0”>
  <SpatialQuery type=”kd−tree” test_visibility=”false” />

  <Common time_step=”0.1” />
  <Helbing agent_scale=”2000” obstacle_scale=”4000” reaction_time=”
      0.5” body_force=”1200”          friction=”2400” force_distance
      =”0.015” />
  <Karamouzas orient_weight=”0.8” fov=”200” reaction_time=”0.4”
      wall_steepness=”2”                     wall_distance=”2”
      colliding_count=”5” d_min=”1” d_mid=”8” d_max=”10” agent_force
      =”4” />

  <AgentProfile name=”group1” >
    <Common max_angle_vel=”360” max_neighbors=”10” obstacleSet=”1”
        neighbor_dist=”5” r=”0.19”    class=”2” pref_speed=”1.04”
        max_speed=”2” max_accel=”5” priority=”0.0”>
      <Property name=”pref_speed” dist=”n” mean=”1.3” stddev=”0.15”
          />
    </Common>
    <Helbing mass=”80” />
    <Karamouzas personal_space=”0.69” anticipation=”8” />
    <ORCA tau=”3.0” tauObst=”0.15” />
  </AgentProfile>

  <AgentGroup>
    <ProfileSelector type=”const” name=”group1” />
    <StateSelector type=”const” name=”Walk” />
    <Generator type=”hex_lattice” anchor_x=”1.5” anchor_y=”0.0”
        alignment=”center” row_direction=”y” density=”1.8” width=”
        4.0” population=”100” rotation=”−90” displace_dist=”n”
                    displace_mean=”0.1” displace_stddev=”0.03” />
  </AgentGroup>

  <ObstacleSet type=”explicit” class=”1”>
    <Obstacle closed=”1” >
      <Vertex p_x=”−20” p_y=”2.0” />
      <Vertex p_x=”20” p_y=”2.0” />
      <Vertex p_x=”20” p_y=”−2” />
      <Vertex p_x=”−20” p_y=”−2” />
    </Obstacle>
  </ObstacleSet>
</Experiment>
```

Listing 2 Behavior specification for a periodic hallway

```
<BFSM>
  <GoalSet id=”0”>
    <Goal type=”AABB” id=”0” min_x=”−20” max_x=”−15” min_y=”−2.0”
        max_y=”2”/>
```

```
  </GoalSet>

  <State name=”Walk” final=”0” >
    <GoalSelector type=”explicit” goal_set=”0” goal=”0” />
    <VelComponent type=”goal” />
  </State>
  <State name=”GoalReached” final=”0”>
    <GoalSelector type=”identity” />
    <VelComponent type=”zero” />
    <Action type=”teleport” dist=”u” min_x=”13.5” max_x=”14” min_y=
        ”−1.5” max_y=”1.5” />
  </State>

  <Transition from=”Walk” to=”GoalReached” >
    <Condition type=”AABB” min_x=”−40” max_x=”−13.5” min_y=”−2.0”
        max_y=”2.0” inside=”1” />
  </Transition>
  <Transition from=”GoalReached” to=”Walk” >
    <Condition type=”auto” />
  </Transition>
</BFSM>
```