Menge: A Modular Framework for Simulating Crowd Movement

Sean Curtis* Andrew Best† Dinesh Manocha‡

University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/Menge/
https://github.com/MengeCrowdSim/Menge

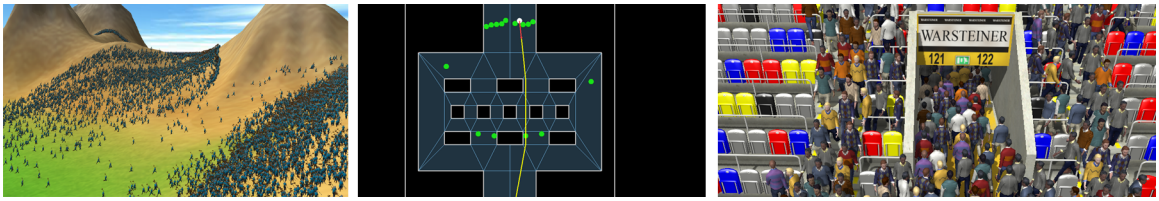


Figure 1: *Three example scenarios simulated using the Menge crowd simulation framework, from left to right: a battlefield with 32,000 agents running across uneven terrain, Menge's interactive display during a global navigation experiment, and a simulation of a human experiment in soccer stand. All of these interactive simulations can be easily generated using the modular architecture and behavior specification of Menge.*

Abstract

We present Menge, a cross-platform, extensible, modular framework for simulating agent 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 can greatly 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.

1 Introduction

Whether for interactive graphics, special effects, or engineering applications, crowd simulation – the simulation of a large number of independent entities, agents, 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 if its purpose is to safely move from one location to another. To solve this subproblem, one could use data structures including, but not limited to, potential fields [Khatib 1986], road maps [Latombe 1991], navigation meshes [Snook 2000], or corridor maps [Geraerts et al. 2008]. 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 other 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. Such a framework would have multiple benefits:

- 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.

*e-mail: seanc@cs.unc.edu
†e-mail: best@cs.unc.edu
‡e-mail: dm@cs.unc.edu

- 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 evaluation As users release their novel subproblem solutions back to the framework, other users of the framework could make direct, meaningful comparisons 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.

To this end, we present *Menge*, a modular, open-source, cross-platform framework for simulating crowd movement, designed to realize all three potential benefits outlined above. Menge is the German word for "crowd", "multitude", or "many". It refers not only to the application domain – crowds – but also to the architecture – a system built as a composition of many *elements*, which correspond to solutions for crowd simulation subproblems. The Menge crowd simulator can be dynamically configured from pre-existing modules, and novel modules can easily be implemented and dynamically plugged into the system. The simulator and high-level agent behaviors are defined by the user through an extendible XML specification. This XML specification allows for rapid development of complex scenarios in which a population of heterogeneous agents engage in diverse activities and move towards disparate goals. We show the importance and efficacy of this modular approach by simulating a set of representative crowd simulation scenarios, illustrating the impact the choice of subproblem solution can have, the expressive range of the XML specification, and the benefits of its extensibility for particular scenarios. With the goals above in mind, Menge provides the follow key benefits:

- Menge can simulate most prior multi-agent microscopic methods providing a strong basis for comparison with new pedestrian models and local navigation algorithms.
- Menge provides a simple framework for mixing and matching solutions to subproblems. Novel simulators can be constructed easily by choosing different components, such as global and local navigation, and spatial acceleration mechanisms.
- Menge supports complex navigation topologies and environments.
- Menge treats agents as independent entities, and parallelizes simulation updates in an agent-parallel manner. Menge is able to simulate tens of thousands of agents at interactive rates on multi-core processors.
- Menge can simulate diverse and complex behaviors and populations through its extensible Behavioral Finite State Machine (BFSM).
- Menge's extensible elements give users freedom to implement new elements according to any arbitrary mechanism they choose. Menge provides simple extensible interfaces for parsing XML, managing shared resources, and interacting with other elements.
- Menge offers a simple, interactive visualizer which displays agent and simulation parameters and realtime and supports exporting trajectories and behaviors to binary format for use with external visualizers and motion synthesizers.

In this paper, we show that a large number of prior crowd movement simulation algorithms can be easily implemented within Menge's framework. Furthermore, we use Menge to develop new crowd simulation algorithms and systems that can model crowd formations, density-dependent behaviors, and movement of airplane passengers. The rest of the paper is organized in the following manner. In Section 2 we describe the underlying abstraction of crowd simulation which Menge assumes and show how it relates to recent work in crowd simulation, in Section 3 we describe the main features of Menge's architecture, in Section 4, we illustrate Menge's compatibility through comparison with existing techniques and implemented example scenarios, and, finally, we offer our concluding thoughts in Section 5.

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 2). This is not a novel abstraction; it has been referred to in previous work [Funge et al. 1999; Ulicny and Thalmann 2002] and is well represented in the crowd simulation literature (although there are notable exceptions). 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, such as psychology, world knowledge, and context. 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 [2005] used *situation calculus* to author a complex train platform scenario . Ulincy and Thalmann [2002] computed high-level behaviors with a combination of rules and behavior finite state machines. Similarly, Bandini et al. [2006] used finite state machines to model complex behaviors with a cellular automata pedestrian model. Paris and Donikian uses 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) [Paris and Donikian 2009]. 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*, requires a *static* plan to achieve the goal. This is most typically associated with *motion planning* [Latombe 1991]. If the goal requires the agent to perform an action at its current location, the motion planning is deferred to the motion synthesis stage. If the goal requires the agent to traverse the simulation domain, then the problem becomes "path planning". An explicit path is not strictly necessary; it merely serves as the basis of computing an instantaneous *preferred velocity* for the agent – the velocity the agent would take towards the goal, if unhindered.

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 [Latombe 1991], navigation meshes [Snook 2000; Oliva and Pelechano 2011], delaunay triangulation

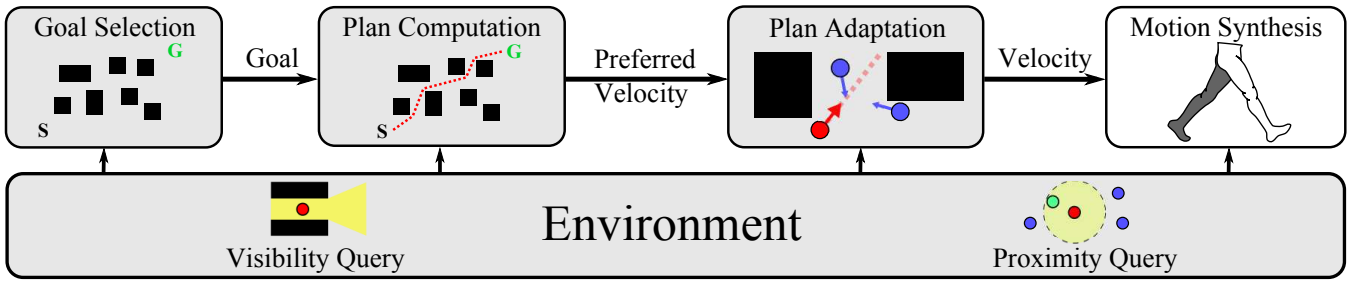


Figure 2: *An abstraction of crowd simulation based on subproblems. First, a goal is selected. Second, a base plan to reach that goal is computed. Third, the plan is adapted to local, unexpected conditions. Finally, motion is synthesized in support of the realized plan. Each subproblem can make queries 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.*

[Lamarche and Donikian 2004], and corridor maps [Geraerts et al. 2008]. 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., [Jaillet and Simeon 2004; Kallman and Mataric 2004; Yang and Brock 2007]).

Another common approach uses potential fields. The simulation domain is discretized and a field is computed that is the gradient of a cost function [Khatib 1986]. No path is explicitly computed. Instead, the resultant vector field provides a direction of "optimal" travel toward the goal. The preferred velocity computed as a solution to this subproblem is provided as input to the next.

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 transform 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 [Schadschneider 2002], social forces [Helbing and Molnár 1995; Pelechano et al. 2007; Karamouzas et al. 2009], vision-based [Ondřej et al. 2010], continuum-based [Narain et al. 2009], velocity-obstacle-based [van den Berg et al. 2011; Pettré et al. 2009], and rule based [Reynolds 1987]. All of these models are compatible with the plan adaptation abstraction.

It is worth noting that there are crowd models which use a different paradigm (e.g., [Treuille et al. 2006; Kapadia et al. 2009]). 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 [Bruderlin and Calvert 1989; Sun and Metaxas 2001], data-driven methods [Multon et al. 1999; Kovar et al. 2002; Heck and Gleicher 2007; Lee et al. 2007], and, for locomotion, foot-step driven methods [van Basten et al. 2011].

2.5 Environmental Queries

Finally, the various subproblems typically need to perform spatial queries in the environment. For example, it is common 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 to the reader to the following resources for visibility queries [Cohen-Or et al. 2003] and proximity queries [Samet 2006].

2.6 Crowd Systems

There is also research in full crowd simulation systems, e.g., Autonomous Pedestrians [Shao and Terzopoulos 2005]. Autonomous Pedestrians, in part inspired by Newell's [Newell 1990] Unified Theories of Cognition, expresses the crowd simulation problem as a composition of conceptual layers. 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–SteerSuite[Singh et al. 2009a] and ADAPT [Kapadia et al. 2014]. We provide a more detailed comparison with these systems in Section 5.1.

3 Menge's Architecture

In this section, we discuss the design philosophy and architecture of Menge. Menge's architecture is primarily focused on facilitating the simulation of agents *moving* through a shared space [1].

3.1 Mathematical Realization

The problem of computing agent trajectories can be thought of as an Initial Value Problem (I.V.P):

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

where $\dot{\mathbf{x}}_i(t)$ or $\mathbf{v}_i(t)$ is the instantaneous velocity of agent i, $\mathbb{S}(t)$ is the simulator *state* at time t, and $\mathbf{V}_i$ is some 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.

[1] 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.)

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 $\mathbb{X} = \bigcup_i \mathcal{X}_i$ is the union of each agent's state space. The minimum agent state space necessary to satisfy the differential equation is $\mathcal{X}_i = [\mathbf{x}_i \ \mathbf{v}_i]^T$, where $\mathbf{x}_i$ and $\mathbf{v}_i \in \mathbb{R}^2$. Menge assumes that simulation is performed in a two-dimensional domain[2]. In practice, particular solutions to the initial value problem require additional per-agent properties which extends 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 I.V.P. abstraction may 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[3].

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, $A_i : S \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$, 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.

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 (1) and as:

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

Menge implements this abstraction. Each subproblem function is implemented by a set of one or more orthogonal *elements* in Menge. A particular crowd simulator can be instantiated by specifying particular elements and their relationships. For example, it is trivial to configure two different simulators such that they use the same solutions to the goal selection and path planning subproblems, but different path adaptation solutions (see Section 4.1 for specific examples). This is how one would perform comparisons between two or more steering algorithms. Menge also introduces an element to provide the visibility and proximity queries.

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 radically 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.

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

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

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 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 useful implementations of 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 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 3. 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 an 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 even can 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 [Ulicny and Thalmann 2002; Bandini et al. 2006]. 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 outgoing 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

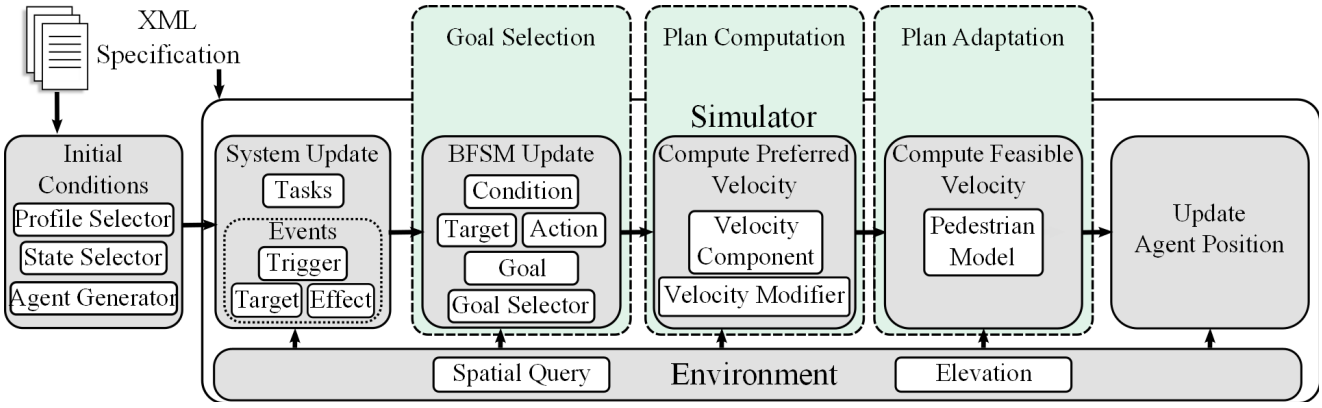


Figure 3: *Menge's computation pipeline. The modular* elements *are shown in white boxes. The green boxes show how the elements relate to the conceptual subproblems. The simulator definition (including initial conditions and BFSM) is given as an XML specification. At each time step, the system updates 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.*

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`. These can be combined or new implementations can be introduced to the system via its plug-in architecture. 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's 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. `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. 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.

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. 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 a `Goal Selector` 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 agent without leading to artificial contention. Menge `Goals` can also be given finite capacity, limiting the number of agents which can simultaneously share 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. This behavior can be modified; the `Goal Selector` can be made "persistent", meaning that the `Goal` assigned the first time 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 [Hutton 2012]. 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*. The relationship between agent and

goal can range from incredibly simple (standing still) to incredibly complex (navigating a maze). Menge is architected in such a way as to easily admit the possibility for selecting the best solution for a given context. 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, 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. [2012], preferred velocity is not actually represented as a simple vector. Instead, Menge allows for preferred velocity to be an *arc* of velocities. The arc represents a space of velocities all of which would cause the agent to travel through a space of locally 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.

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 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 adaptation stages of the simulation pipeline. 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., [Golas et al. 2013; He and van den Berg 2013]); 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 account for and validate required models. Menge's initial release includes several models including two velocity-obstacle-based models and several force-based models. Additional models are forthcoming.

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 elements.

The `Spatial Query` element provides an interface to perform visibility and proximity queries. Implementations of novel spatial query algorithms and data structures can easily 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.

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).

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 3). 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 work is only performed once. Alternatively, a `Task` can be explicitly instantiated by the user in the XML specification. For example, Menge includes a `Task` for computing and visualizing density of the crowd.

Menge provides the basis of a complex event system. An

event is uniquely defined by three elements: `EventTrigger`, `EventEffect`, and `EventTarget`. 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 provide an important mechanism in conjunction with the BFSM. The BFSM determines the progression of an agent's behavior through a well-defined path in the BFSM. Events provide a mechanism to alter agent states in an asynchronous, arbitrary fashion.

`EventTriggers` 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 `EventTrigger`'s arbitrary internal state. `EventTargets` 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. `EventEffects` encode the actual effect of the event when triggered. `EventEffects` can include changing b-State parameters of agents, disabling transitions, and terminating the simulation.

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 `AgentGenerator`, `ProfileSelector`, `StateSelector` 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 `ObstacleSet` element.

The `AgentGenerator` is responsible for generating a number of agents and assigning them positions (the agent's a-state). Menge provides several implementations to facilitate construction simulation scenarios 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.

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 specify either a single value, or 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 (mean: 1.3 m/s, standard deviation 0.1 m/s). A `ProfileSelector` assigns a user-defined profile to each agent. The assignment criteria is, as with all Menge elements, arbitrary. They could be based on position in the simulation, random assignment, etc. Profiles and `ProfileSelector` elements permit the user to quickly create heterogeneous crowds. The populations can easily be varied to facilitate experimentation.

The `StateSelector` is similar to the `ProfileSelector`. The `StateSelector` 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. This allows the encoding of multiple disparate behaviors; the behaviors of different categories of agent in a scene (police, pedestrians, etc.) would consist of disconnected sub-graphs in the BFSM. 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 `StateSelector`. The `StateSelector` facilitates the creation of behavioral categories.

`ObstacleSets` specify the impassable "walls" in the simulation. These may be the boundaries of an office building, or hazards which are activated dynamically. `ObstacleSets` 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.

4 Application and Evaluation

In this section we examine specific examples which illustrate the strengths and properties of Menge. 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 other facets of 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 at `http://gamma.cs.unc.edu/Menge/`.

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 4(a)). In this example, we vary the `Pedestrian Model` implementation between a velocity-obstacle model [van den Berg et al. 2011], a simple social-force model [Helbing et al. 2000], and a predictive social-force model [Karamouzas et al. 2009]; all other aspects of the simulation are fixed. We can observe the differences in behavior due to the `Pedestrian Model` easily (as illustrated by the sample trajectories shown in Figure 5).

Obstacle Course: The obstacle course experiment compares global planning algorithms. The agents shown in Figure 6 must traverse the scene from top to bottom. This 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 [Singh et al. 2009b]. 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.

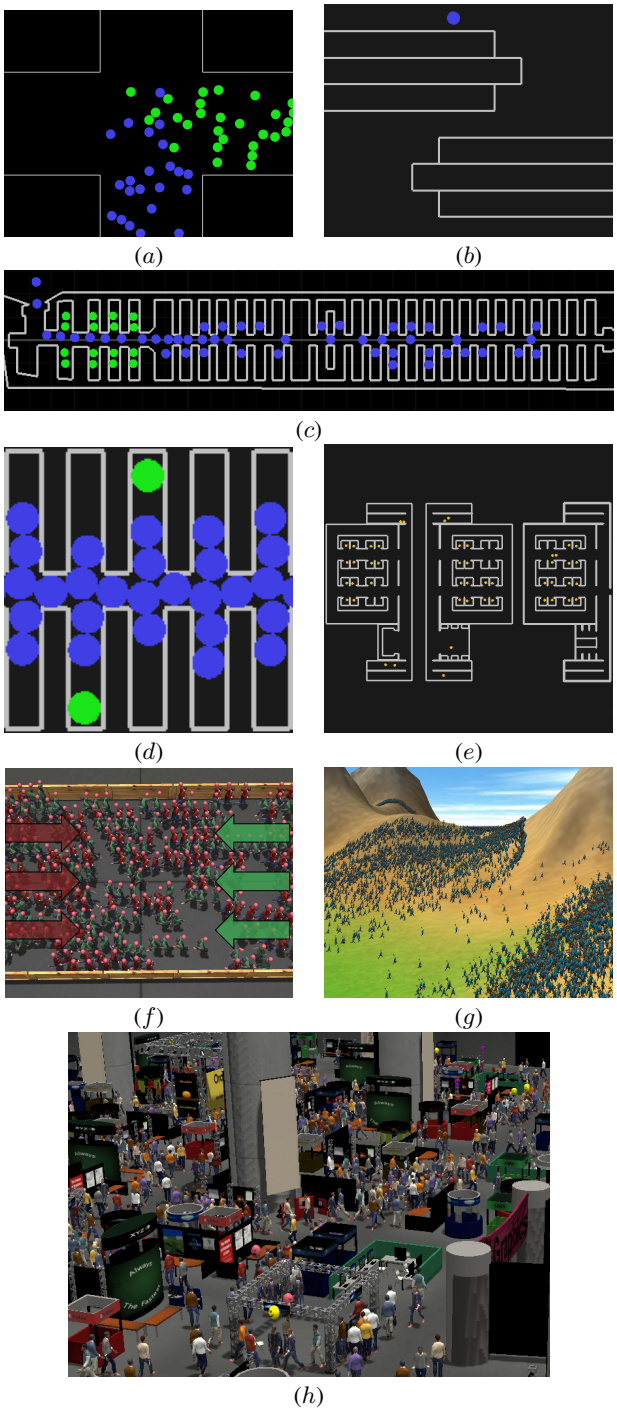


Figure 4: *Images from the various prototype scenarios in Menge. (a) A cross flow highlighting pedestrian model comparisons. (b) A benchmark translated from SteerBench XML. (c) Airplane loading using random goal selection. (d) Agents (green) waiting for the aisle to clear using a custom transition in the airplane. (e) Agents work at desks and perform other activities in a three-story office building. (f) General Adaptation Syndrome algorithm simulation. (g) A battle scene showing 32,000 agents moving across complex terrain at interactive simulation rates. (h) The trade show scene demonstrating agents moving to and judging exhibits.*

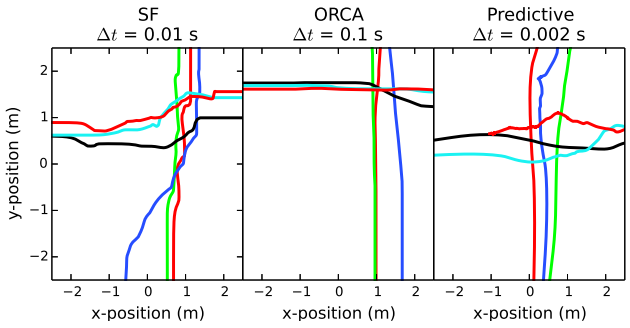


Figure 5: *Trajectories plotted for three different pedestrian models in the Cross Flow scenario: a velocity-obstacle model (ORCA), a simple social force based model (SF), and a predictive forces model (Predictive). With all other simulation elements the same, these trajectories illustrate differences in the model behaviors.*

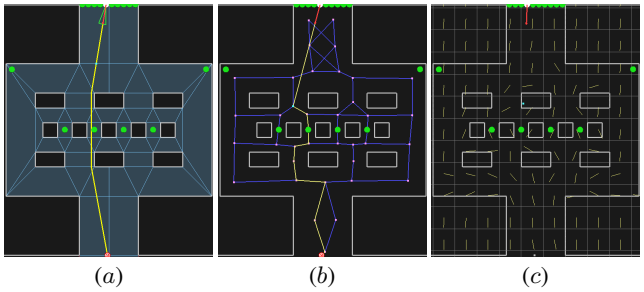


Figure 6: *Visualization of three different global navigation methods applied to the Obstacle Course scenario. The green discs are agents; the yellow line represents the path computed for each algorithm. In the case of the guidance field, each cell's direction vector is shown in yellow. (a) the navigation 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`

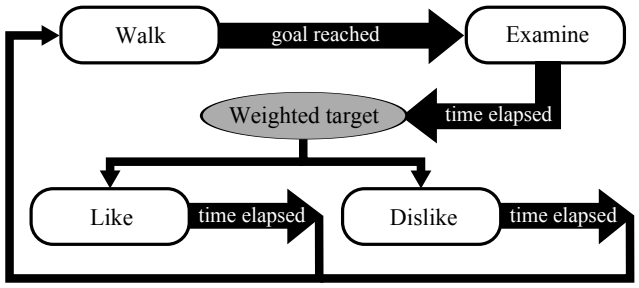


Figure 7: *An illustration of the BFSM used in the trade show scenario. The white boxes represent FSM-states, the black arrows represent transition* `Conditions`*, the grey circle is a transition* `Target`*. Agents walk to an exhibit. When they reach the exhibit they enter the "Examine" state and stay 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.*

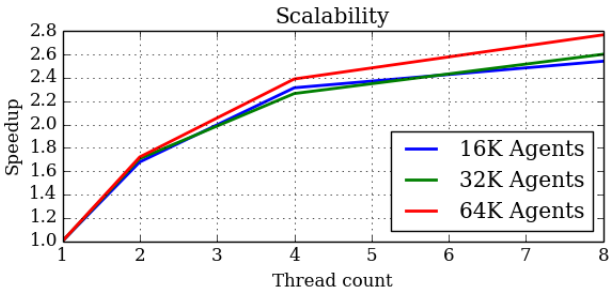


Figure 8: *The results of a scalability experiment for Menge. The Battle scene was simulated using 16,000, 32,000, and 64,000 agents, respectively. The simulation used direct to goal navigation with a terrain sensitive* `VelocityModifider` *and ORCA for local navigation. The average frame computation time was measured based on the number of threads. The speed up over a single thread is shown. The use of a rectangle placement* `AgentGenerator` *makes the scaling of the scene simple. We only change parameter for number agents to be created in the XML.*

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. Our 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 7. 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`. 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 24,000 agents. The terrain is defined by a height field. The `Elevation` 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 greatest population and provides an opportunity to show how Menge scales with population. Figure 8 reports the performance as we varied the population[4]. 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

[4]A simple task when using parametric `AgentGenerator` elements.

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

Figure 9: *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 [Burghardt et al. 2012]. (b) A rendered screenshot of the experiment replicated in Menge. (c) The complex 3D navigation mesh used for the Stadium scenario. The navigation mesh provides elevation information, navigation, and provides a spatial query structure.*

(illustrated in Figure 9). 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` 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. 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 [2012]. Essentially, as stress accumulates, people respond by exhibiting more aggressive behaviors. The authors modeled the accumulation of stress and used work by Guy et al. to model personality changes [2011]. 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 were able to implement this in Menge with a custom `Action` which applies the so-called aggres-

(a) (b) (c)

Figure 10: *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.*

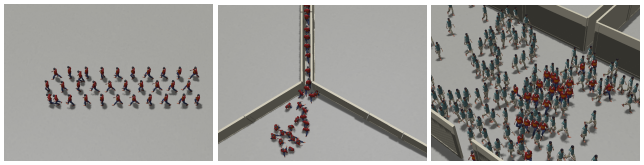
sion displacement on agents. We assign the `Action` to a stress-inducing FSM-state and include a transition which causes the agent to periodically re-enter the state–shorter periods model a higher rate of stress accumulation, longer periods, a slower accumulation rate. We reproduced one of Kim et al.'s experiments: two groups of agents moving in anti-parallel directions in a wide corridor (see Figure 4f. 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 [Gu and Deng 2013; Ju et al. 2010; Zhang et al. 2013]. As a representative sample, we implemented the approach of Gu and Deng to illustrate how easily formations can be introduced into Menge [2013]. This approach defines a formation via a canonical collection *sentinel* points–transform-invariant positions which most importantly define the formation. At each time step, the algorithm maps the canonical definition to simulation space and assigns agents in the formation, in a prioritized manner, first to the sentinel points and then to the remaining points. This transformation and assignment process can account for moving formations, changing formations, and even changing formation populations. Please see the original authors' work for the exact details [Gu and Deng 2013].

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 10 illustrates some of the results using these new elements. In the video, we show one example in which a single group of agents changing formations 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. 2014a], 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 incredibly constrained environment.

Ped-Air exploits Menge's `GoalSelector` element to model passenger seat assignment and 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) (b) (c)

Figure 11: *Visualization of the "Formation Stress" scenario. This brief experiment conveys the simplicity of combining previously incompatible algorithms to produce a novel simulator. This experiment combines results from stress modeling, formations, and Fundamental Diagram adherence [Kim et al. 2012; Best et al. 2014b; Gu and Deng 2013]. (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.*

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, it is common for some passengers to 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 commonly observed phenomenon in crowd behaviors; as crowds get denser, they get slower [Weidmann 1993]. Best et al. propose an algorithm (DenseSense), based on Menge, which successfully reproduces this behavior [2014b]. 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. In this example, we have combined three separate research results into a single scenario: GAS, formations, and density-dependent behaviors. By having them 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 11).

5 Analysis and Comparisons

5.1 System Comparisons

Menge shares a common philosophical foundation with SteerSuite [Singh et al. 2009a]. Like Menge, SteerSuite is an open-source framework presented with the intention to serve as a common platform for research. It includes a suite of benchmarks and analytical tools to measure performance. The primary difference between SteerSuite and Menge is in the level of extensibility. SteerSuite's purpose, as the name implies, is concerned with *steering* algorithms. Menge seeks to make *all* aspects of crowd simulation extensible.

The ADAPT framework [Kapadia et al. 2014] also addresses the challenges of crowd movement simulation. Like Menge, ADAPT uses a modular architecture to simulate agents moving through a shared environment. Unlike menge, ADAPT focuses primarily on the animation of virtual agents and motion synthesis as it relates to crowd simulation. ADAPT represents simulation agents as fully articulated characters and offers a variety of modular controls for animation. The modular components principal to the navigation system, however, are less flexible than Menge's equivalent elements. In addition, ADAPT uses a behavior tree as opposed to a BFSM. The two mechanisms are essentially equivalent; each has strengths and weaknesses but the same expressive capacity.

5.2 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 allows combinations of algorithmic techniques that were not possible previously. It is an established practice to produce algorithms which target a particular subproblem in crowd simulation without consideration of how those algorithms fit into a rich and robust pedestrian simulator. 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 a common core framework upon which to build.

Menge is open-source, cross-platform, and publicly available at `http://gamma.cs.unc.com/Menge/`. 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 the supporting code or binaries for the community's benefit and future comparisons.

5.3 Limitations and Future Work

Menge has some limitations. First, it currently only allows one mechanism for generating high level behaviors–the BFSM. Behavior trees are a common structure in game AI [Rabin 2008]. 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. Third, 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 [Shum et al. 2008] and [Hyun et al. 2013], 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 parallelism 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 functionality in mind, so that it can be introduced in the future with a minimum of difficulty.

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.

References

BANDINI, S., FEDERICI, M., MANZONI, S., AND VIZZARI, G. 2006. Towards a methodology for situated cellular agent based crowd simulations. *Engineering societies in the agents world VI*, 203–220.

BEST, A., CURTIS, S., KASIK, D., SENESAC, C., SIKORA, T., AND MANOCHA, D. 2014. Ped-air: a simulator for loading, unloading, and evacuating aircraft. In *7th International Conference on Pedestrian and Evacuation Dynamics*.

BEST, A., NARANG, S., CURTIS, S., AND MANOCHA, D. 2014. Densesense: Interactive crowd simulation using density-dependent filters. In *Symposium on Computer Animation*.

BRUDERLIN, A., AND CALVERT, T. W. 1989. Goal-directed, dynamic animation of human walking. In *Proc. of SIGGRAPH '89*, 233–242.

BURGHARDT, S., KLINGSCH, W., AND SEYFRIED, A. 2012. Analysis of flow-influencing factors in mouths of grandstands. In *Pedestrian and Evacuation Dynamics*.

COHEN-OR, D., CHRYSANTHOU, Y., SILVA, C., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (July-Sept.), 412–431.

CURTIS, S., SNAPE, J., AND MANOCHA, D. 2012. Way portals: Efficient multi-agent navigation with line-segment goals. *Proc. of Symposium on Interactive 3D Graphics and Games*, 15–22.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proc. of SIGGRAPH*, 29–38.

GERAERTS, R., KAMPHUIS, A., KARAMOUZAS, I., AND OVERMARS, M. 2008. Using the corridor map method for path planning for a large number of characters. In *Motion in Games*. Springer, Heidelberg, 11–22.

GOLAS, A., NARAIN, R., AND LIN, M. 2013. Hybrid long-range collision avoidance for crowd simulation. *Proc. of Symposium on Interactive 3D Graphics and Games (I3D)*, 29–36.

GU, Q., AND DENG, Z. 2013. Generating freestyle group formations in agent-based crowd simulations. *IEEE Computer Graphics and Applications 33*, 1, 20–31.

GUY, S. J., KIM, S., LIN, M. C., AND MANOCHA, D. 2011. Simulating heterogeneous crowd behaviors using personality trait theory. In *SCA '11*, 43–52.

HE, L., AND VAN DEN BERG, J. 2013. Meso-scale planning for multi-agent navigation. In *Proc. IEEE Int. Conf. on Robotics and Automation - ICRA*.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, 129–136.

HELBING, D., AND MOLNÁR, P. 1995. Social force model for pedestrian dynamics. *Physical Review E 51*, 5 (May), 4282–4286.

HELBING, D., FARKAS, I., AND VICSEK, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 487–490.

HUTTON, A. 2012. London bridge station, the role of ped modelling: Pedestrian modelling and design development. In *6th International Conference on Pedestrian and Evacuation Dynamics*.

HYUN, K., KIM, M., HWANG, Y., AND LEE, J. 2013. Tiling motion patches. *IEEE Trans. Vis. Comput. Graph 19*, 11, 1923–1934.

JAILLET, L., AND SIMEON, T. 2004. A PRM-based motion planning for dynamically changing environments. In *Proc. IEEE RSJ Int. Conf. Intell. Robot. Syst.*, vol. 2, 1606–1611.

JU, E., CHOI, M. G., PARK, M., LEE, J., LEE, K. H., AND TAKAHASHI, S. 2010. Morphable crowds. *ACM Trans. Graph 29*, 6, 140.

KALLMAN, M., AND MATARIC, M. 2004. Motion planning using dynamic roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 5, 4399–4404.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 215–223.

KAPADIA, M., MARSHAK, N., SHOULSON, A., AND BADLER, N. I. 2014. ADAPT: The agent development and prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics 20*, 7 (July), 1035–1047.

KARAMOUZAS, I., HEIL, P., VAN BEEK, P., AND OVERMARS, M. H. 2009. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games*, Springer, vol. 5884 of *Lecture Notes in Computer Science*, 41–52.

KHATIB, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res. 5*, 90–98.

KIM, S., GUY, S. J., MANOCHA, D., AND LIN, M. C. 2012. Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. *ACM Symposium on Interactive 3D Graphics and Games*, 55–62.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Trans. Graph. 21*, 3, 473–482.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3, 509–518.

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Springer, Heidelberg.

LEE, K. H., CHOI, M. G., HONG, Q., AND LEE, J. 2007. Group behavior from video: a data-driven approach to crowd simulation. In *Symposium on Computer Animation*, 109–118.

MULTON, F., FRANCE, L., CANI-GASCUEL, M.-P., AND DEBUNNE, G. 1999. *Computer animation of human walking: a survey*, vol. 10. 39–54.

NARAIN, R., GOLAS, A., CURTIS, S., AND LIN, M. C. 2009. Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph. 28*, 122:1–122:8.

NEWELL, A. 1990. *Unified theories of cognition*. Harvard University Press, Cambridge, MA, USA.

OLIVA, R., AND PELECHANO, N. 2011. Automatic generation of suboptimal navmeshes. In *MIG*, Springer, J. M. Allbeck and P. Faloutsos, Eds., vol. 7060 of *Lecture Notes in Computer Science*, 328–339.

ONDŘEJ, J., PETTRÉ, J., OLIVIER, A.-H., AND DONIKIAN, S. 2010. A synthetic-vision based steering approach for crowd simulation. In *Proc. SIGGRAPH*, 123:1–123:9.

PARIS, S., AND DONIKIAN, S. 2009. Activity-driven populace: A cognitive approach to crowd simulation. *Computer Graphics and Applications, IEEE 29*, 4, 34–43.

PELECHANO, N., ALLBECK, J., AND BADLER, N. 2007. Controlling individual agents in high-density crowd simulation. In *Symposium on Computer Animation*, 99–108.

PETTRÉ, J., ONDŘEJ, J., OLIVIER, A.-H., CRETUAL, A., AND DONIKIAN, S. 2009. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Symposium on Computer Animation*, 189–198.

RABIN, S. 2008. *AI Game Programming Wisdom 4 (AI Game Programming Wisdom (W/CD))*, 1 ed. Charles River Media.

REYNOLDS, C. 1987. Flocks, herds and schools: A distributed behavioral model. In *Proc. of SIGGRAPH*.

SAMET, H. 2006. *Foundations of MultiDimensional and Metric Data Structures*. Morgan Kaufmann.

SCHADSCHNEIDER, A. 2002. Cellular automaton approach to pedestrian dynamics - theory. *Pedestrian and Evacuation Dynamics*, 75–86.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *Symposium on Computer Animation*, 19–28.

SHUM, H. P. H., KOMURA, T., SHIRAISHI, M., AND YAMAZAKI, S. 2008. Interaction patches for multi-character animation. *ACM Trans. Graph 27*, 5, 114.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of the 2nd International Workshop on Motion in Games*, 158–169.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. Steerbench: a benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds 20*, 5-6, 533–548.

SNOOK, G. 2000. Simplified 3D movement and pathfinding using navigation meshes. In *Game Programming Gems*. Charles River, Hingham, Mass., ch. 3, 288–304.

SUN, H. C., AND METAXAS, D. N. 2001. Automating gait generation. In *Proc. of ACM SIGGRAPH*, 261–270.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *Proc. of ACM SIGGRAPH*, 1160–1168.

ULICNY, B., AND THALMANN, D. 2002. Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum 21*, 4, 767–775.

VAN BASTEN, B. J. H., STUVEL, S. A., AND EGGES, A. 2011. A hybrid interpolation scheme for footprint-driven walking synthesis. *Graphics Interface*, 9–16.

VAN DEN BERG, J., GUY, S. J., LIN, M., AND MANOCHA, D. 2011. Reciprocal n-body collision avoidance. In *Inter. Symp. on Robotics Research*, 3–19.

WEIDMANN, U. 1993. Transporttechnik der fussgaenger. Tech. Rep. 90.

YANG, Y., AND BROCK, O. 2007. Elastic roadmaps: globally task-consistent motion for autonomous mobile manipulation. In *Proc. Robot. Sci. Syst.*, 279–286.

ZHANG, P., LIU, H., AND DING, Y.-H. 2013. Crowd simulation based on constrained and controlled group formation. *The Visual Computer*, 1–14.

5.4 Funding Acknowledgment

The work at UNC Chapel Hill has been supported by NSF award 1305286, and a grant from the Boeing Company.

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 12 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`)[5].

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`).

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 12(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 12.

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.

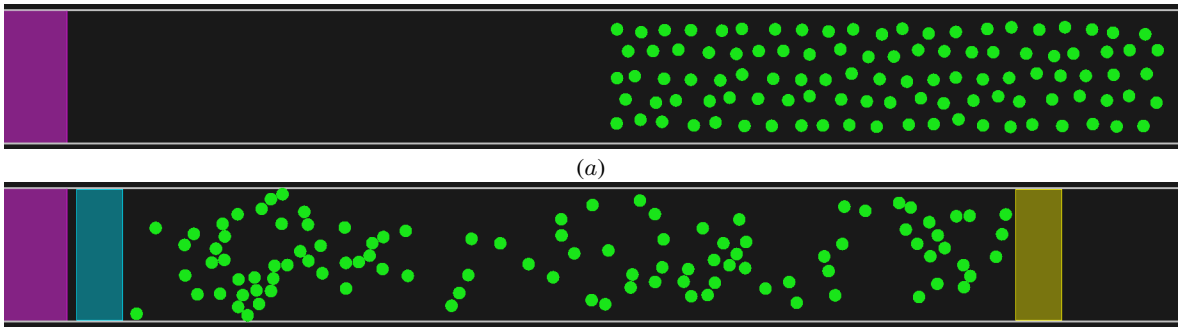
Line 12 This FSM-state's `VelocityComponent` is the `zero` type. Every agent in this state will have the zero preferred velocity.

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 12(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

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



(*a*)

(*b*)

Figure 12: *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.*

box on the left in Figure 12(*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>
```