403 Poyntz Ave., Suite B
Manhattan, KS 66502, USA
Phone: +1-785-770-8511
Email: support@thunderheadeng.com
Web: https://www.thunderheadeng.com

Pathfinder

Pathfinder Technical Reference Manual

Version: 2020-4
Last Modified:

Table of Contents

Disclaimer

Thunderhead Engineering makes no warranty, expressed or implied, to users of Pathfinder, and accepts no responsibility for its use. Users of Pathfinder assume sole responsibility under Federal law for determining the appropriateness of its use in any particular application; for any conclusions drawn from the results of its use; and for any actions taken or not taken as a result of analyses performed using these tools.

Users are warned that Pathfinder is intended for use only by those competent in the field of egress modeling. Pathfinder is intended only to supplement the informed judgment of the qualified user.

The software package is a computer model that may or may not have predictive capability when applied to a specific set of factual circumstances. Lack of accurate predictions by the model could lead to erroneous conclusions. All results should be evaluated by an informed user.

All other product or company names that are mentioned in this publication are tradenames, trademarks, or registered trademarks of their respective owners.

Throughout this document, the mention of computer hardware or commercial software does not constitute endorsement by Thunderhead Engineering, nor does it indicate that the products are necessarily those best suited for the intended purpose.

Acknowledgements

This work was partially funded by a Small Business Innovative Research (SBIR) grant by the United States National Science Foundation (NSF).

We would like to thank Rolf Jensen and Associates for their assistance with testing and other suggestions that helped guide the development of the simulator.

In addition, we would like to thank all the beta testers who contributed feedback on the web forums and via email.

Chapter 1. Overview

Pathfinder is an agent-based egress simulator that uses steering behaviors to model occupant motion. It consists of three modules: a graphical user interface, the simulator, and a 3D results viewer.

Pathfinder provides two primary options for occupant motion, SFPE mode and Steering mode.

SFPE mode (Chapter 5)

Implements the concepts in the SFPE Handbook of Fire Protection Engineering (SFPE 2016). This is a flow model, where walking speeds are determined by occupant density within each room and flow through doors is controlled by door width.

Steering mode (Chapter 6)

Based on the idea of inverse steering behaviors. Steering behaviors were first presented in Craig Reinolds' paper "Steering Behaviors For Autonomous Characters" (Reinolds 1999) and later refined into inverse steering behaviors in a paper by Heni Ben Amor (Amor et al. 2006). Pathfinder's steering mode allows more complex behavior to naturally emerge as a byproduct of the movement algorithms - eliminating the need for explicit door queues and density calculations.

Pathfinder is included in the NIST Review of Evacuation Models (Kuligowski, Peacock, and Hoskins 2010), but the most current set of features and capabilities may not be reflected in that document.

1.1. Example Problem IMO Test 10

In the following discussions, it is often useful to have an example with which to illustrate particular points. One frequently referenced example is Test 10 from the International Maritime Organization (IMO) (IMO 2016).

This test problem represents a cabin corridor section as shown in Figure 1, the cabins are populated as indicated. The population consists of males 30-50 years old with a minimum walking speed of `0.97 m/s`, a mean speed of `1.30 m/s`, and a maximum speed of `1.62 m/s`. There is no delay in response and the walking speeds are distributed uniformly between the minimum and maximum to the `23` occupants. The passengers in cabins 5 and 6 are assigned the secondary exit; all the remaining passengers use the main exit. The expected result is that the allocated passengers move to the appropriate exits.

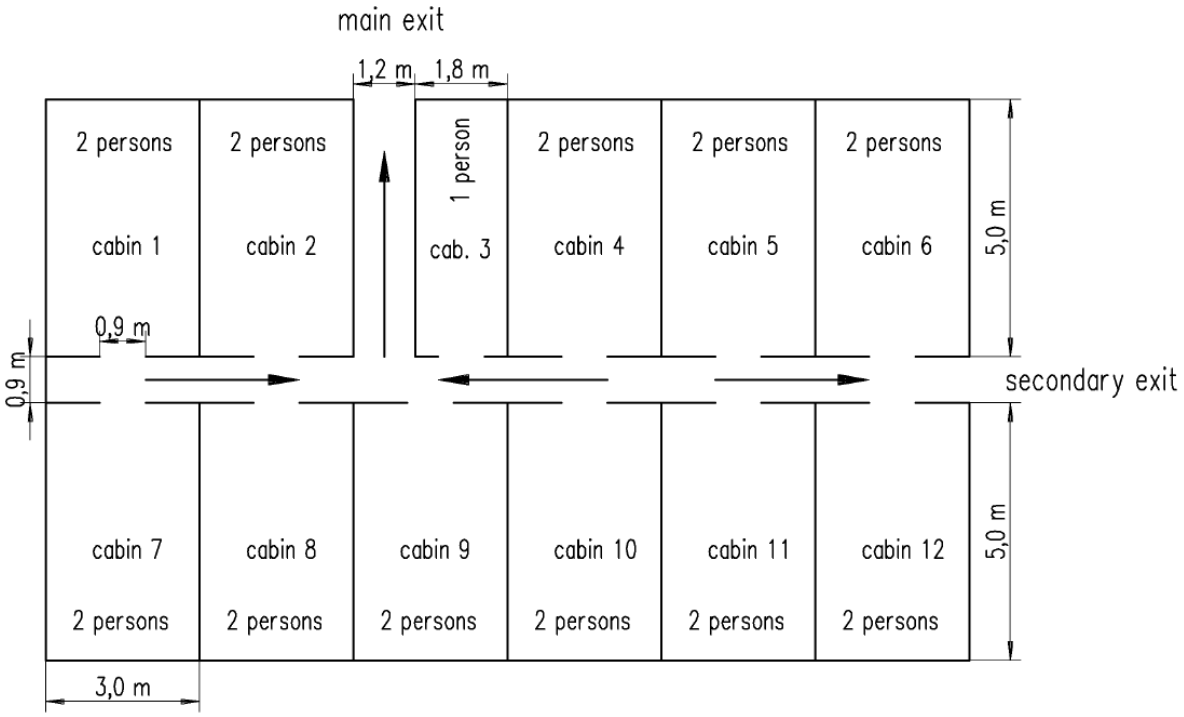

Figure 1. Cabin area (from IMO, 2002)

The display of occupant movement, Figure 2, does show that occupants have selected the assigned exits. In this display, the paths of all occupants are displayed, with selected occupants and their paths highlighted. For the steering mode analysis, all occupants exited the corridor in `18.0` seconds.

The results for SFPE mode are illustrated in Figure 3. In SFPE mode, the passengers form a queue at the main exit and the flow through this door controls the exit time. For the SFPE analysis, all occupants exited in 21.2 seconds. In SFPE mode, occupants can overlap in space during movement and when the queue forms.

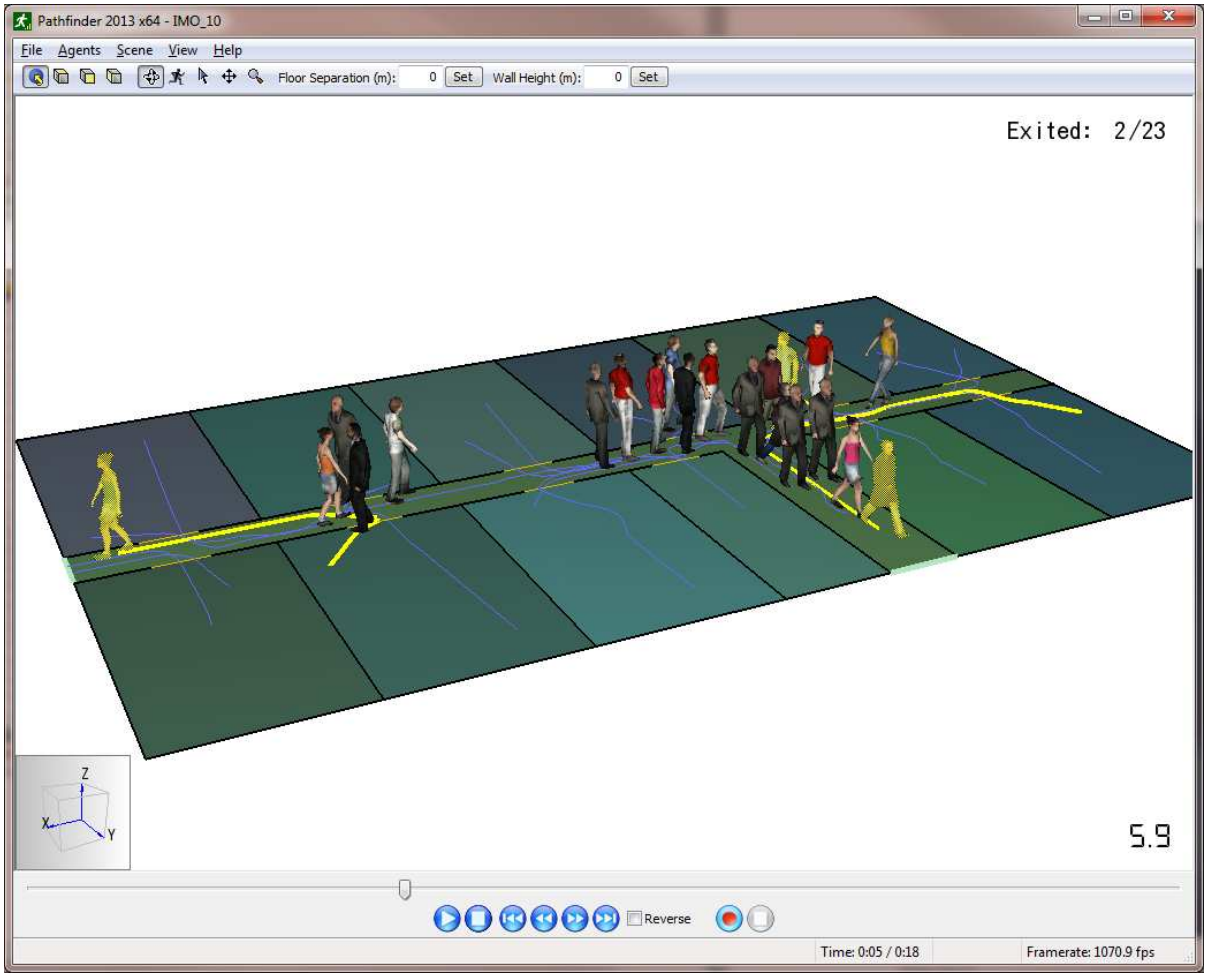

Figure 2. Steering mode results for IMO 10 problem showing occupant movement, note how highlighted occupants move to their assigned exits.

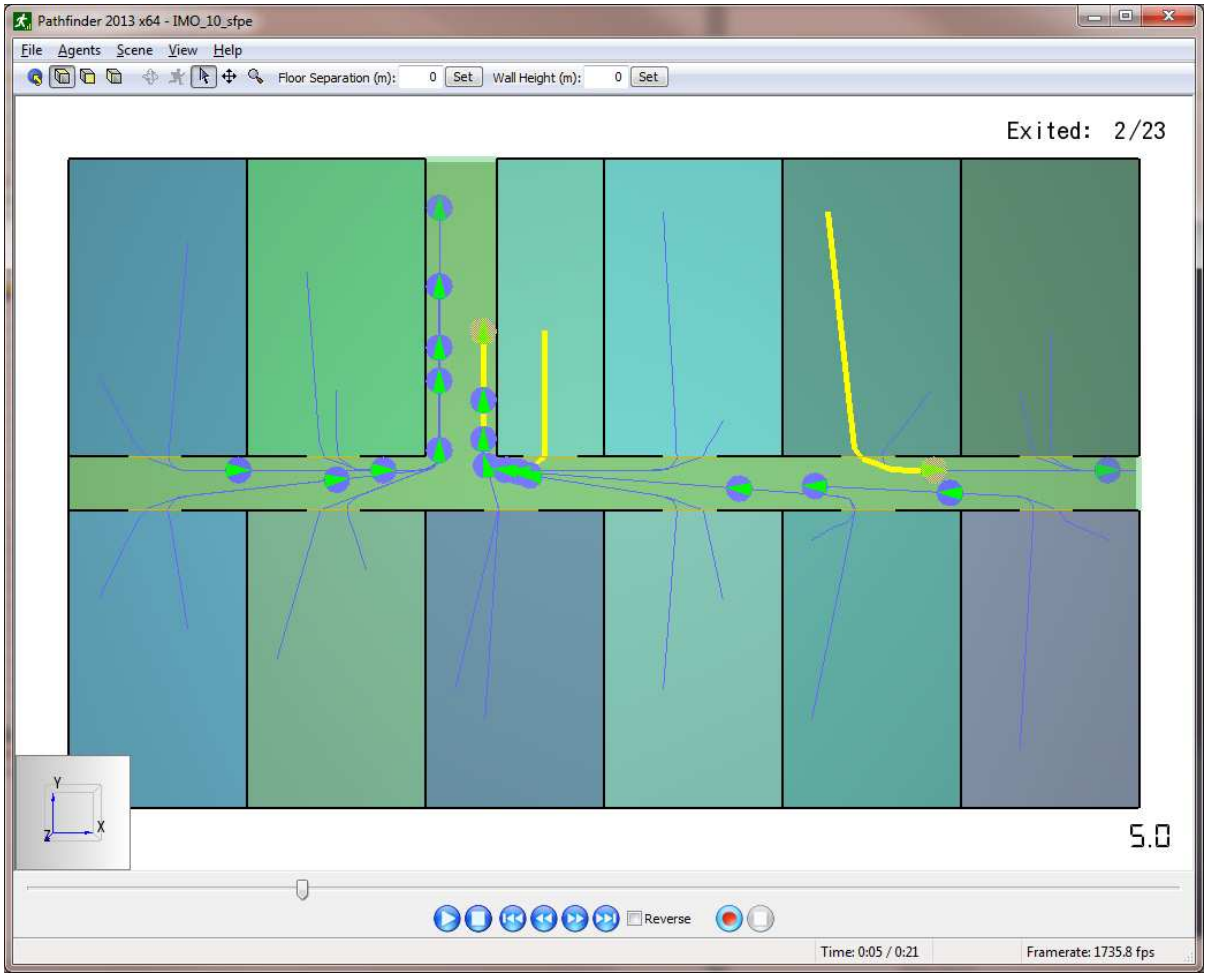

Figure 3. SFPE mode result for IMO 10 problem, note that multiple occupants can occupy the same space.

Chapter 2. Geometry

Pathfinder uses a 3D geometry model (Figure 4). Within this geometric model is a navigation mesh defined as a continuous 2D triangulated surface referred to as a "navigation mesh". The navigation mesh is an irregular one-sided surface represented by adjacent triangles. Occupant motion takes place within the boundaries of this navigation mesh.

Figure 4 shows a townhouse model and the corresponding navigation mesh in Figure 5. Pathfinder supports drawing or automatic generation of a navigation mesh from imported geometry – including Fire Dynamics Simulator (FDS) files, PyroSim files, and a collection of other common CAD file formats.

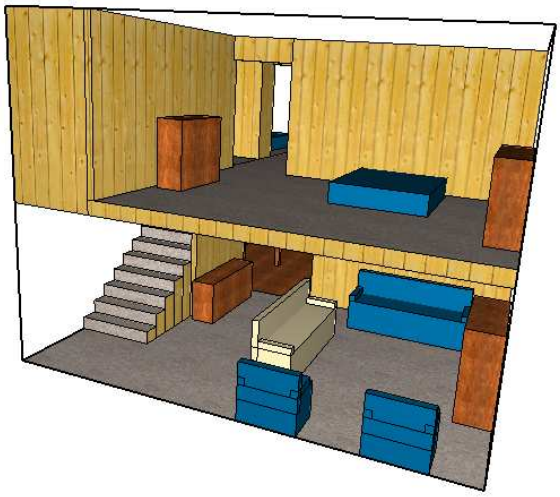

Figure 4. 3D geometry

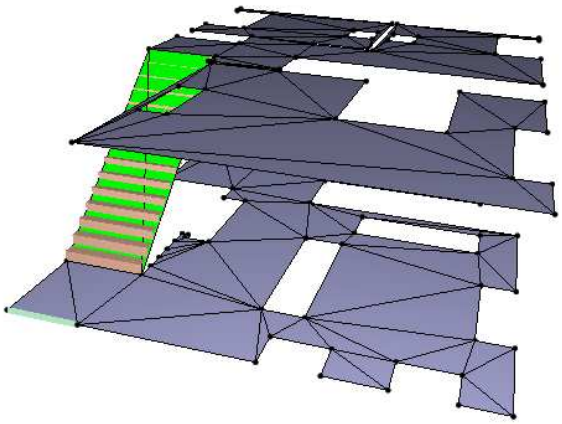

Figure 5. Extracted navigation mesh

As can be seen in Figure 5, obstructions in Pathfinder are represented implicitly as gaps in the navigation mesh. Since occupants can only travel on the navigation mesh, this technique prevents the overhead of any solid object representation from affecting the simulator. When the navigation mesh is generated by importing geometry, any region of the mesh blocked by a solid object is automatically removed. For overhead obstructions, the mesh generator considers any obstruction within `1.8 meters (6 feet)` of the floor to be an obstacle.

2.1. Geometry Subdivision

The navigation geometry is organized into rooms of irregular shape. Each room has interior (optional) and exterior _boundaries_ that cannot be crossed by an occupant.

Travel between two adjacent rooms is through connecting *doors*. A door that does not connect two

rooms and is defined on the exterior boundary of a room is an *exit door*. There can be multiple exit doors in a model. When an occupant enters an exit door in SFPE mode, they are queued at the door and removed at the flow rate defined by SFPE. Occupants that enter an exit door in reactive steering mode are removed from the simulation immediately.

Figure 6 illustrates these concepts for the IMO Test 10 problem. The rooms (and corridors) are shaded different colors. Doors from individual rooms to the corridor (just another room in the model) are indicated by a thick orange line. Exit doors are indicated by a thick light green line. Occupants are shown by the blue dots. Superimposed on the geometry is the navigation mesh.

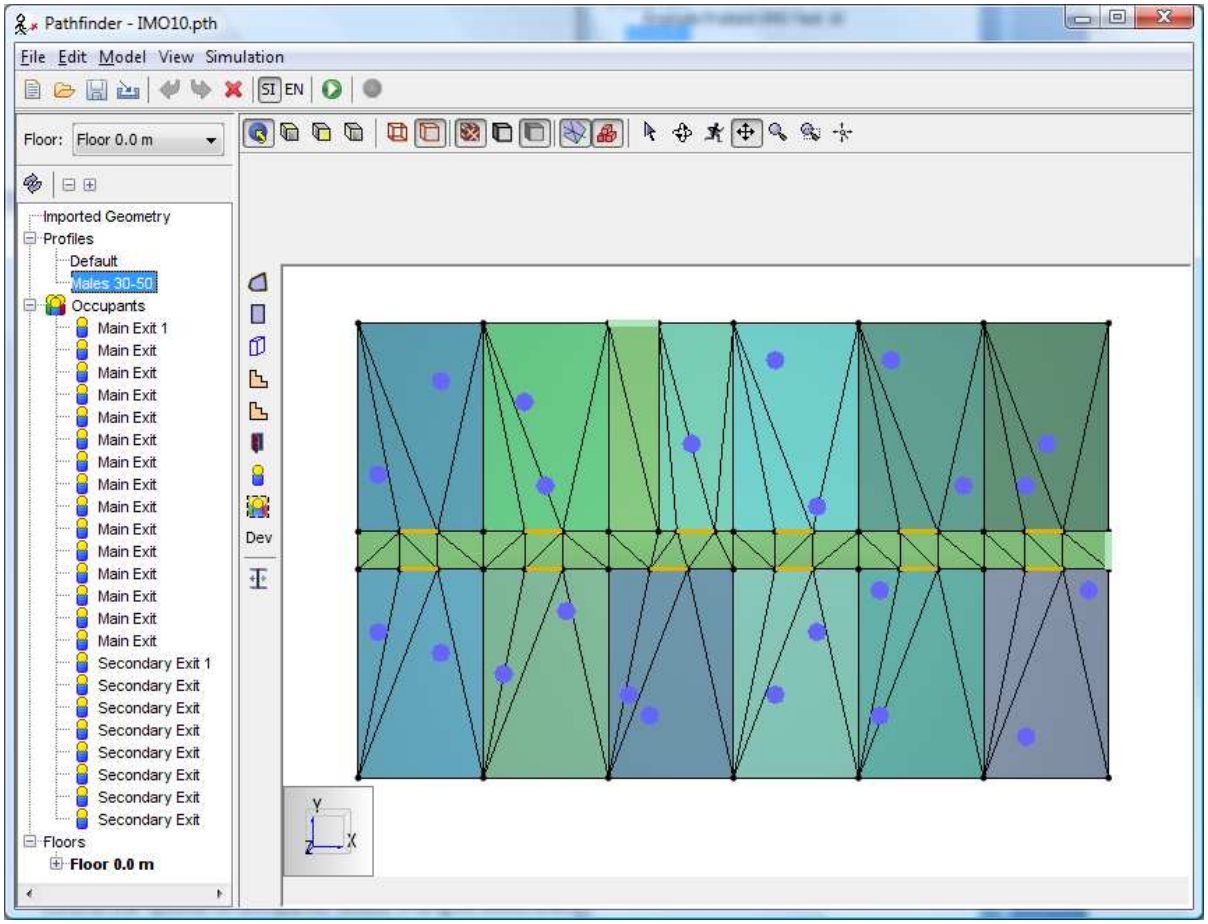

Figure 6. Rooms, doors, exits and the navigable mesh in the IMO Test 10 problem

Any location on the navigation mesh can be categorized as one of four terrain types: open space, stairs, doors and exit. Ramps and rooms are both classified as open space. Each terrain type has an effect on the behavior of occupants on that section of the mesh.

2.1.1. Rooms

The open space within a Room provides no explicit constraints on movement. Rooms created in the user interface using the room drawing tools are all considered open space having level

terrain, even if rotated so that they have slope. In SFPE mode, the maximum walking speed of occupants becomes a function of the occupant density in the room.

NOTE	While a room may be rotated such that it resembles a ramp, it is still a room and is considered to be level terrain in the speed calculations. The only way to create a ramp in the user interface is to use one of the ramp drawing tools.

2.1.2. Stairs

Stairways connect rooms on different levels. They denote areas where the maximum occupant velocity is controlled by an alternate calculation specific to stairways. The specific velocity calculation is given in the stairway section for each simulator mode.

At the top and base of each stairway, there are two doors. While the user can edit the width and activation events for the doors, the doors cannot be directly moved or deleted independently of the stair. These doors connect the stairway mesh to the adjacent rooms' meshes and function identically to ordinary connecting doors.

Like doors, stairs can normally be traversed in either direction, but they can also be marked as one-way. The simulator models this by making both doors on the ends of the stair one-way.

Each stair has associated step rise and step run properties, which are settable in the user interface. This rise/run does not have to match the geometric slope of the stair. This is important as it relates to the calculation of an occupant’s speed on stairs as discussed in Section 5.2.

2.1.3. Ramps

Ramps are created and represented very similarly to stairs in a Pathfinder model, but they are treated very differently in the simulation. Like stairs, ramps have two doors at either end and can be made to be one-way. Unlike stairs, however, ramps do not affect the speed of occupants when using the default SFPE ramp speed calculations. When using a custom ramp speed function, the input slope is geometric and cannot be entered by the user.

2.1.4. Doors (Connecting)

Doors connect two adjacent rooms together. In SFPE mode they act as the main flow control mechanism, as discussed in Section 5.3, but in steering mode, doors merely record the flow between rooms for results viewing unless explicitly set to limit flow.

Normally, occupants can travel through doors in either direction; however, in the user interface the door can be marked as one-way. This limits occupants to travelling through the door in only the indicated direction unless the occupant’s profile dictates otherwise.

2.1.5. Doors (Exit)

Exits are a special case of doors that mark building exits. They will be colored a bright green color to indicate an exit door instead of an interior connecting door which will be a tan color.

Chapter 3. Behaviors and Goals

Each occupant has a *behavior* assigned to them in the user interface. A *behavior* dictates a sequence of *goals* that the occupant must achieve in the simulation.

There are two main types of goals in Pathfinder:

Idle goals

Goals in which an occupant must wait at a location until an event occurs, such as a time-interval elapses or an elevator reaches a discharge floor.

Seek goals

Goals in which an occupant moves toward a destination, such as a waypoint, room, elevator, or exit.

Instantaneous goals

Goals which occur in one timestep.

3.1. Idle Goals

When occupants idle, they wait until an event occurs. While the occupant is waiting in SFPE Mode, they stand still until the event occurs. While the occupant is waiting in Steering Mode, they use *separation* to maintain a distance from other occupants (see Section 6.4.2).

Because the occupant may move in Steering Mode, they are assigned a containment area that depends on the previous seek goal in the occupant's behavior. If the occupant leaves this area because of separation, they create and use a temporary *seek goal* to return to the area.

The areas are defined as follows:

- If the previous seek goal was a *waypoint*, the occupant tries to stay in the radius of the waypoint.
- If the previous seek goal was a room (including an elevator), the occupant tries to stay in the room, away from the doors, allowing other occupants to enter.
- If there was no previous seek goal, the occupant can move anywhere along the mesh.

3.2. Seek Goals

When an occupant seeks, they are trying to find a destination on the mesh. The occupant uses *path planning*, *path generation*, and *path following* to reach the destination as discussed in Chapter 4.

3.3. Goals

The following are the types of goals in Pathfinder. Some have a corresponding *behavior action* in the user interface, while others may be created as sub-goals of a particular behavior action.

Assist Occupants (Seek)

The occupant joins an assisted evacuation team and assists clients, see Section 8.1.

Change Behavior (Instant)

Instructs an occupant to change their Behavior to a new Behavior picked randomly from a Behavior distribution.

Change Profile (Instant)

Instructs an occupant to change to a different Profile picked randomly from a Profile distribution.

Detach from Assistants (Instant)

Causes an assisted client to detach from their assistants.

Fill Room (Idle)

Causes the agent to seek a location in the room away from all active doors, see Room-filling. This goal is not explicitly added to a Behavior; it follows the Goto Room and Wait action and happens while they are waiting.

Goto Elevator (Seek/Idle)

An elevator or set of elevators is defined that the occupant should use. This goal is implemented in Pathfinder through a combination of a room-seek goal and an idle goal. As discussed in Chapter 9, there is a virtual pickup room representing the elevator at each floor to which the elevator attaches. The room-seek portion of the elevator goal points to one or more of these virtual rooms. The virtual rooms are selected based on the location of the previous seek goal. If the previous seek goal is attached to the elevator on the same floor, the elevator's room on this floor is selected as the target room. If not, the next floor down is tested. This continues until the bottom floor is reached. If no elevator connection is found, the search continues with the next floors up from the previous seek goal. This continues until an elevator pickup room is found that connects to the previous seek goal. Once an occupant enters a virtual elevator pickup room, they enter an idle state and wait. Once the elevator reaches the discharge floor, the elevator control system changes the occupant's room to the virtual discharge room, and the occupant finishes the *Goto Elevator* goal.

Goto Exit (Seek)

A door or set of doors is defined that the occupant seeks. The goal is reached once the occupant crosses one of the exit doors. Additionally, the occupant is removed from the simulation.

Goto Room (Seek)

A room or set of rooms is defined that the occupant should seek. This goal is reached once the occupant crosses a door leading into one of the rooms.

Goto Waypoint (Seek)

A point is defined on the mesh along with a radius. The occupant attempts to reach the point. The waypoint is reached once the occupant is within the radius of the point.

Wait (Idle)

The occupant enters an idle state and waits until a timer elapses.

Wait Until (Idle)

The occupant enters an idle state and waits until a specified value of simulation time.

Wait for Assistance (Idle)

The occupant stands still until all *attached occupant positions* are filled by assistants. See Section 8.2 for more information.

In the user interface, an occupant may be assigned a **Goto Refuge Rooms** behavior action. There is no corresponding goal in the simulator, however. For unassisted agents, this merely becomes a *Goto Rooms* goal.

For assisted clients, however, they are assigned the following sequence of goals:

1. *Goto Rooms* – this tells the client to go to the refuge rooms.
2. *Fill Room* – see Room-filling below.
3. *Detach from Assistants* – this allows assistants to detach and continue helping other clients.

3.3.1. Room-filling

In steering mode, when occupants are instructed to go to a room and wait, the occupants will try to fill the room to ensure other occupants may also enter it.

Room filling may be triggered by any of the following scenarios:

- The occupant's behavior has a **Goto Room** goal and then a **Wait** goal.
- The occupant's behavior has a **Goto Elevator** goal.
- The occupant's behavior was assigned a **Goto Refuge Rooms** action in the user interface.

In order to fill a room, occupants go through a series of steps:

1. Obtain a door distance map.
2. Determine an ideal seek direction to move the occupant toward the furthest unoccupied location away from a set of doors.
3. Decide whether to actually move in the ideal seek direction.

3.3.2. Door Distance Map

The first step of room-filling is to acquire a *door distance map* that indicates how far away a door is for any point in the room. This map is generated by sub-dividing the room into smaller triangles as shown in Figure 7. For each vertex of the sub-divided triangle, a distance value is generated that is the minimum distance to a set of doors. A color-mapped version of this distance map is shown in Figure 8.

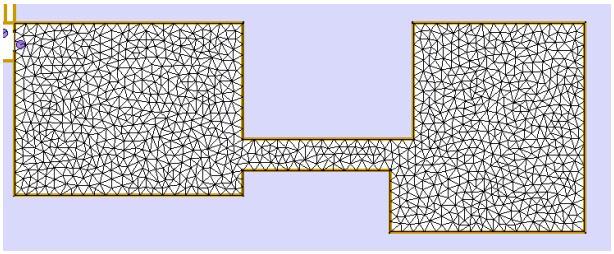

Figure 7. Sub-divided room for door distance map

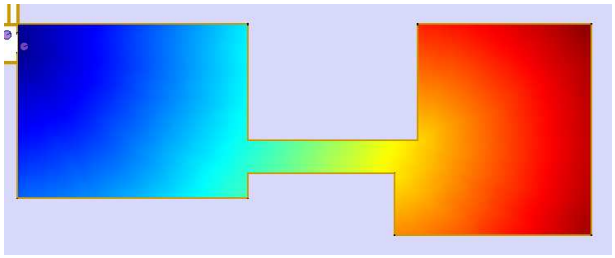

Figure 8. Door distance map

The set of doors used to generate the distance map varies per-occupant based on whether the occupant can move in the room. For assisted occupants who cannot move on their own and will be left in the room, the set includes all doors attached to the room. For all other occupants, the set only includes doors that are active. An active door is one that has occupants waiting to enter on the other side or that has had an occupant pass through into the current room in the past 15 seconds. The occupant will periodically check to see if the door distance map has changed and update their decisions accordingly.

3.3.3. Ideal Seek Direction

Once the occupant has obtained the door distance map, the occupant determines an ideal seek direction.

To do this, the occupant creates sample directions that are 30° apart from each other covering a full 360°. For each sample direction, the occupant samples the distance map to find the maximum door distance that can be obtained in that direction and how far it is to that location. Then the occupant checks to see if they will collide with other occupants in that direction and how far it is to the collision. The occupant then limits the distance that can be travelled in that direction to the minimum of the distance to an occupant collision and distance to the maximum door distance.

The occupant chooses the sample direction that will give the farthest distance from a door according to the door distance map.

3.3.4. Decide Whether to Move

Once the ideal seek direction has been obtained, the occupant determines whether to actually move in that direction or simply stay where they are.

If the occupant is assisted, cannot move on their own, and will be left in the room, they will always move in the ideal seek direction. Otherwise, the occupant will only move if there are other occupants within 1 meter and who are on the negative side of a plane. The plane is determined by the occupant’s location and ideal seek direction.

Chapter 4. Pathfinding

When an occupant has a destination to seek set by a Goal, they need a plan for how to reach the destination, a path to follow, and a way to follow the path while accounting for dynamic obstacles along the path, such as other occupants.

There are two modes that Pathfinder can use for simulating the movement of occupants through the model.

- SFPE Mode implements the flow-based egress modeling techniques presented in the *SFPE Handbook of Fire Protection Engineering* (SFPE 2016) and the *SFPE Engineering Guide: Human Behavior in Fire* (SFPE 2019).
- Steering Mode uses a combination of steering mechanisms and collision handling to control how the occupant follows their seek curve. These mechanisms allow the occupant to deviate from the path while still heading in the correct direction toward their goal.

4.1. Path Planning

Path planning is the process of determining a plan for moving toward a destination.

Given an occupant seeking a destination, there may be multiple paths to reach the destination, each with differing lengths, numbers of occupants along the way, and various hazards. A naïve path planning approach to choosing a route would be to take the shortest route. This may not be the fastest or best route to the destination for a particular occupant, for this they need to find the path out of a room in the least amount of time. In Pathfinder this is solved by the Locally Quickest method.

The occupant uses the following steps to plan a path.

1. Generate a list of *local targets*. By default, the local targets include the doors attached to the occupant's current room that can lead to the seek goal.

   > **NOTE** A door leads to a seek goal if the shortest path from that door to the goal goes immediately outside the current room or a path exists from the door to the goal that does not pass through any rooms more than once and does not go through the current room (see Section 4.1.3).

   If the seek goal is in the current room, it is added to the list of local targets.

2. Choose a local target based on local and global knowledge of the model and occupant preferences. This is discussed in Section 4.1.2.

3. Move toward the local target using *path generation* and *path following*, periodically repeating these steps until the final seek goal is reached.

4.1.1. Locally Quickest

Locally Quickest is the path planning approach used in Pathfinder to solve this problem. It plans the route hierarchically, using local information about the occupant's current room and global knowledge of the building. It is assumed that an occupant knows about all doors in their current room as well as the queues at those doors. It is also assumed that the occupant knows how far it is from one of those doors to the current destination (seek goal). Locally quickest then uses this information to choose a door in the current room based on a calculated cost of that door. A path is then generated to the door, which the occupant can follow.

4.1.2. Door Choice

As mentioned in step 2 of the path planning process, the occupant chooses a local target by calculating a cost for the target and choosing the target with the lowest cost. The cost for each target is based on multiple criteria and the occupant's preferences.

The criteria are as follows:

Current room travel time, t_{lt}

The time it would take the occupant to reach the target at maximum speed, ignoring all other occupants.

Current room queue time, t_q

If the target is a door, this is an estimate of the time the occupant will have wait in the queue at the door based on the occupant's position in the queue and the flow rate of the door. If the target is not a door, the queue time is 0. The flow rate through doors is calculated using a bi-quad low-pass filter with a cut-off frequency of .05 Hz. Unless there is counterflow at the door, the flow rate as seen by the occupant is clamped so that it will never be less than *min_flowrate_factor * door_nominal_flow*. *min_flowrate_factor* is settable in the user interface as **Minimum Flowrate Factor**, and *door_nominal_flow* is the nominal flow rate of the door as calculated by SFPE.

NOTE	The flow rate filter cutoff frequency is not currently settable in the user interface.

Global travel time, t_{gt}

The time it would take the occupant to travel from the target to the current seek goal at the occupant's maximum speed, ignoring all other occupants. If the target is the current seek goal, this time is 0.

NOTE	If two targets have global travel times within 10% of the longer time, the global travel time is treated as the lower of the values. This causes occupants to prefer the closer door if two targets have similar global travel times.

Distance travelled in room, d_t

The distance the occupant has travelled since entering the current room.

Each occupant also has a set of door choice preferences that are settable in the user interface.

These preferences are:

Current Room Travel Time Cost Factor, k_{lt}

A cost factor associated with the current room travel time.

Current Room Queue Time Cost Factor, k_q

A cost factor associated with the current room queue time.

Global Travel Time Cost Factor, k_{gt}

A cost factor associated with the global travel time.

Current Door Preference, p

A value that gives preference to the occupant's most recently chosen target. This value also helps to prevent occupants from frequently switching local targets.

Current Room Distance Penalty, k_{dd}

A doubling distance that is turned into a cost factor. This factor causes the travel time costs to increase as the occupant travels further in a room. It has the effect of causing occupants to prefer doors with shorter overall distances to shorter times the further they travel in a room. This is a simplistic way to model fatigue and helps to dampen the frequency at which occupants switch local targets.

The cost of a particular target is then calculated as follows:

$$C_{target} = C_l + C_g$$

$$C_g = p_d k_{gt} t_{gt}$$

$$C_l = \max\left(p_d k_{lt} t_{lt} | k_{qh} k_q t_q\right)$$

$$p_d = e^{k_d d_t}$$

$$k_d = \frac{\log 2}{k_{dd}}$$

In the equation above, k_{qh} is set to `1.0 - p` for the most recently chosen target and `1.0` for all other targets.

Each occupant chooses a door using this technique when they first enter a room. The second door choice in the room is randomized per-occupant between 0 to 1 second later. The third and subsequent door choices occur at intervals of 1 second from the second door choice.

NOTE This interval is currently not settable in the user interface.

4.1.3. Backtrack Prevention

Occupants are only aware of queue sizes and door flow rates in their current room. When they enter a new room, knowledge about the last room is replaced by knowledge about the current room. Without any sort of backtrack prevention in place, large queues could lead to occupants walking back-and-forth between two rooms, potentially for long periods of time (until the previous room emptied).

In Pathfinder, once an occupant manages to exit a room using a particular exit door, they are committed to that routing decision using the following rules:

1. The next local door the occupant selects may not lead back into any previous rooms. If this rule eliminates all options (e.g. the occupant went through an unplanned door), then
2. Backtrack prevention is disabled, the occupant can choose from any valid local door.

4.2. Path Generation

Once a local target has been chosen through path planning, a path is needed to reach the target. Pathfinder uses the $A*$ search algorithm (Hart, Nilsson, and Raphael 1968) and the triangulated navigation mesh. The resulting path is represented as a series of points on edges of mesh triangles. These points from $A*$ create a jagged path to the occupant's goal.

To smooth out this jagged path, Pathfinder then uses a variation on a technique known as *string pulling* (Johnson 2006). This re-aligns the points so the resulting path only bends at the corner of obstructions but remains at least the occupant's radius away from those obstructions.

Examples of these generated points, called *waypoints,* are shown in Figure 9. This example shows the projected path of an occupant in a simple rectangular room. The occupant is standing in the lower-left corner and plans to exit out the lower-right corner. The navigation mesh is shown by the thin lines that form triangles inside the rectangular area. An obstruction prevents the occupant from walking straight to the exit. The planned path of the occupant is shown as the dark line and the waypoints are shown as circles. A waypoint is generated for each edge that intersects the path.

Once these waypoints are found, Pathfinder removes intermediate points that lie between two others in a straight line. This creates a series of waypoints only where the direction of travel will change.

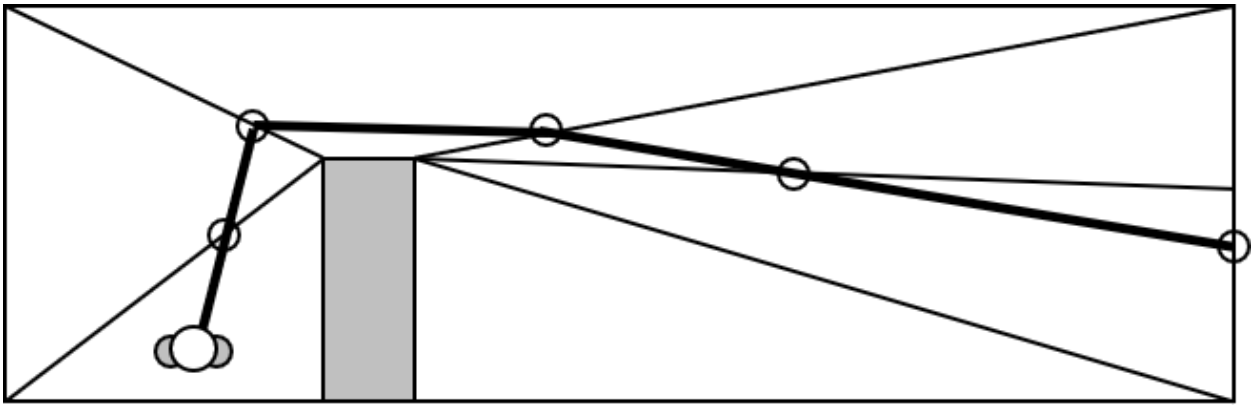

Figure 9. An occupant's planned path with waypoints shown

4.3. Path Following

Once a path is generated, the occupant needs a way to follow the waypoints.

The occupant performs the following:

1. Two waypoints are tracked:
    a. a *current waypoint* that is initially the furthest waypoint on the path that defines a bend in the path
    b. a *next waypoint* that defines the next bend in the path.
2. The occupant checks if the next waypoint should become the current. This is determined by checking if the occupant crossed an infinite line connecting the current waypoint with the next waypoint. If the line is crossed, the next waypoint becomes the current and a new next waypoint is determined.
3. The occupant checks for the need to re-path. Occupants must re-path if they cannot see a straight line to their current waypoint or if it is time to re-evaluate the current door choice according to locally quickest.
4. A *seek curve* is generated to define the desired motion. In SFPE Mode, this curve is merely a straight line segment from the current position to the current waypoint. In Steering Mode, this is a quadratic B-spline using the current position, the current waypoint, and a control point that is projected back along the direction from the current waypoint to the next waypoint.
5. The occupant attempts to move along the tangent to the current seek curve. This movement is strongly influenced by the movement mode (SFPE or Steering) and is discussed in the next sections.

The following images show the paths and waypoints for the IMO Test 10 problem for both SFPE (Figure 10) and Steering (Figure 11) modes. The green lines indicate the current seek curves for each occupant. The red lines and points indicate future paths and waypoints. Notice the straight seek curve in the SFPE mode as compared to the spline used in steering mode.

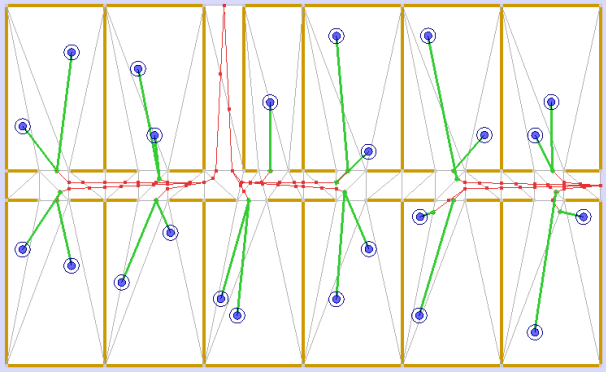

Figure 10. Paths and Waypoints in SFPE Mode

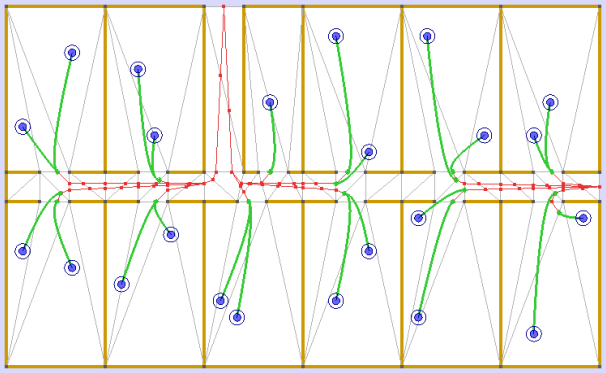

Figure 11. Paths and Waypoints in Steering Mode

Chapter 5. SFPE Mode

Pathfinder provides the option to calculate motion in an *SFPE Mode*. This mode implements the flow-based egress modeling techniques presented in the *SFPE Handbook of Fire Protection Engineering* (SFPE 2016) and the *SFPE Engineering Guide: Human Behavior in Fire* (SFPE 2019). The SFPE calculation as described in the handbook is a flow model, where walking speeds and flow rates through doors and corridors are defined.

In Pathfinder, navigation geometry can be grouped into three types of components; rooms, doors, and stairs. Rooms are open space on which occupants can walk. Doors are flow limiters that connect rooms and stairs. Stairs can be thought of as specialized rooms in which the slopes of the stairs limit the speed of the occupants. There is no specialized corridor type as in the SFPE guide. Instead, corridors are modeled as rooms with doors on either end. In this manner, corridors are handled in the same manner as rooms, with the flow being controlled by the doors.

In SFPE mode, multiple occupants can occupy the same point on the navigation surface.

5.1. SFPE Mode Parameters

In SFPE Mode, the following parameters are used.

Max Room Density *(0.0 < D_{max}, default=3.55 pers/m^2)*

This parameter controls how many occupants will be allowed to enter a room via doors and stairways. Pathfinder uses room density to determine movement speed and door flow rate. When occupants queue at doors, they will not be able to leave the queue on their turn unless doing so will keep the density in the next room below this value.

Boundary Layer *(0.0 <= BL)*

This value controls the effective width of every door in the simulation – including doors associated with stairs. The effective width of a door is $W - 2 * BL$ where W is the actual width of the door. The effective width of a door controls the rate at which occupants can pass through the door.

> **NOTE** This same boundary layer is used when calculating the room density, as defined in section Section 5.2.2.

Door Flow Rates at High Density, Use a Calculated Specific Flow *(on/off, default=on)*

This flag controls the calculation of door specific flow with respect to density. If this flag is enabled, specific flow for doors is calculated based on the occupant density in adjacent rooms. This calculation is explained in Section 5.3.

Door Flow Rates at High Density, Always Use Max Specific Flow *(on/off, default=off)*

This flag controls the calculation of door specific flow with respect to density. If this flag is enabled, doors always use maximum specific flow.

5.2. Velocity

The velocity (v) at which an occupant moves depends on several factors, including the occupant's maximum velocity (v_{max}) specified in the user interface, the type of terrain being travelled on, speed modifiers and constants associated with the terrain, and occupant density in the current room.

5.2.1. Base Speed

The occupant's base speed (v_b) is defined as a function of density, terrain, and a speed fraction curve based on the SFPE fundamental diagram. It does not take terrain speed modifiers or constants into account.

$$v_b = v_{max} * v_f(D) * v_{ft}$$

Where:

v_{max} is the occupant's maximum speed as entered in the user interface as **Speed**.

$v_f(D)$ is a speed fraction as a function of density as follows:

$$v_f(D) = \begin{cases} 1, & D < .55\, pers/m^2 \\ \max\left[v_{fmin}\frac{1}{.85}(1-.266D)\right], & D \geq .55\, pers/m^2 \end{cases}$$

v_{fmin} is a *minimum speed fraction* as defined in the user interface (default=.15), and D is the occupant density in the current room.

v_{ft} is a speed fraction that depends on the type of terrain being traversed by the occupant.

It is defined as:

$$v_{ft} = \frac{k}{1.4}$$

For level terrain (rooms) and ramps, $k = 1.40 m/s$.

For stairs, k depends on the step slope of the stairway. The SFPE handbook defines k only for a limited set of known step slopes as follows:

Table 1. SFPE Handbook table of k values

Stair Riser (inches)	Stair Tread (inches)	k
7.5	10.0	1.00
7.0	11.0	1.08
6.5	12.0	1.16
6.5	13.0	1.23

Pathfinder uses this information to determine k values for any stairs by constructing a piece-wise linear function that maps step slope to k values using these known data points.

The step slope of a stair is defined as:

$$\text{step slope} = \frac{\text{stair riser}}{\text{stair tread}}$$

For step slopes above `.75` (the maximum in the table), the values are extrapolated down to a minimum k of `.034`. This ensures that very steep stairs do not cause occupants to become excessively slow. For step slopes below `.5` (the minimum in the table), k is linearly interpolated to `1.4` at a step slope of `0` (while not realistic, this would correspond to a flat stairway). This produces a k function as shown in Figure 12.

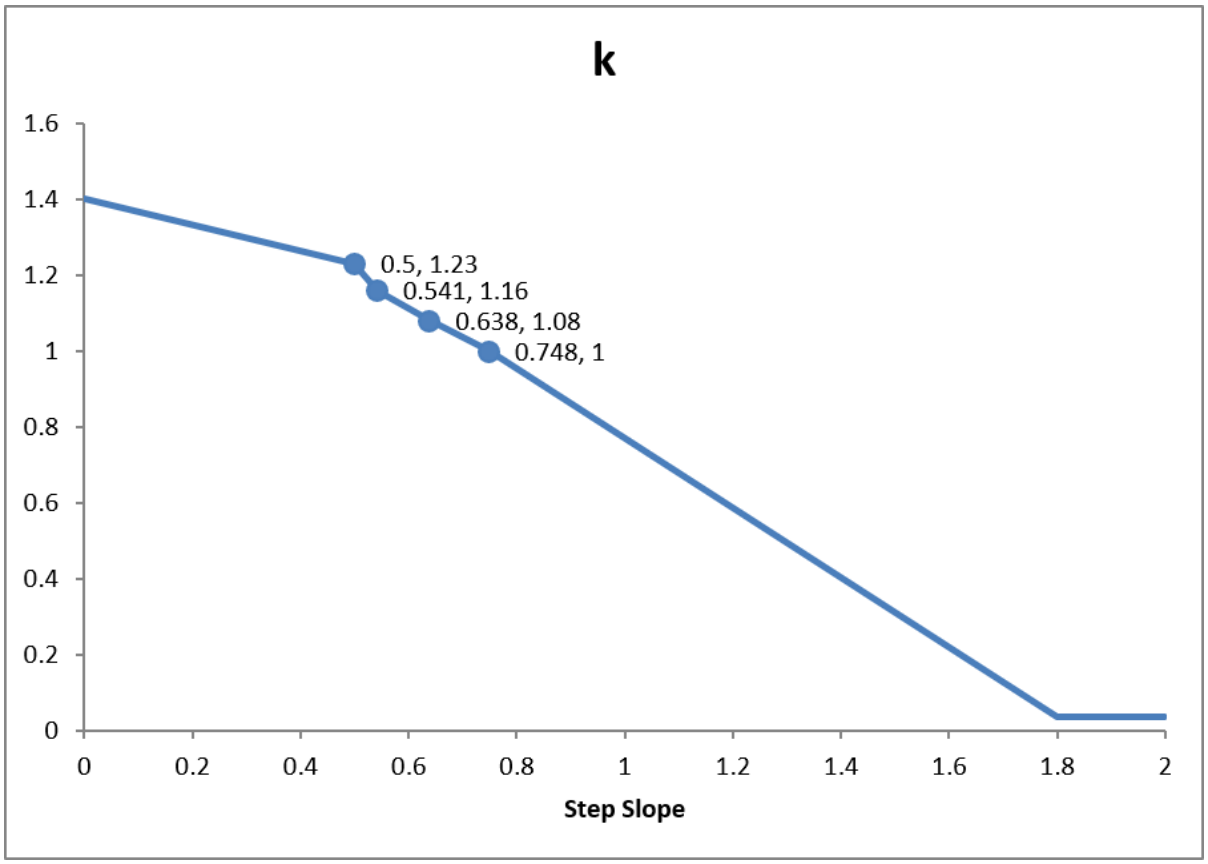

Figure 12. k as a function of step slope

5.2.2. Density

In SFPE mode, density is considered uniform throughout a single room. It is calculated as follows:

$$D = \frac{n_{pers}}{A_{room} - A_{blayer}}$$

n_{pers} is the number of occupants in the room

A_{room} is the area of the room

A_{blayer} is the area of the boundary layer, which is calculated by multiplying the total length of the boundary edges in the room by the *Boundary Layer* as set on the **Behavior** tab of the **Simulation Parameters** dialog under **Door Flow Rate** when **SFPE** mode is selected.

5.2.3. Speed Modifiers and Constants

Egress components, such as rooms, stairs, and ramps, can be assigned speed modifiers and speed constants in the user interface, which can be used to emulate environmental effects, such as smoke, and specialized navigational geometry such as escalators and moving walkways. By default, all egress components have a speed modifier of 1.

When an occupant enters an egress component with a speed modifier, the occupant's speed on that component is calculated as follows:

$$v = k_v + v_b$$

where k_v is the speed modifier for the component, and v_b is the occupant's base speed on the component.

If the component has a speed constant instead of a speed modifier, the occupant's speed depends on the occupant's profile parameter, *Walk on Escalators*, and the speed constant value.

If *Walk on Escalators* is turned on or the speed constant's value is 0, the occupant's speed is:

$$v = v_c + v_b$$

Where v_c is the speed constant for the component.

Otherwise, the occupant's speed is:

$$v = v_c$$

5.3. Movement through Doors

When using Pathfinder in *SFPE Mode*, the occupant flow rate through the door is specified by the SFPE guidelines. This is implemented using a delay timer that controls how quickly occupants are allowed to pass through the door. This timer is initially set at zero. When an occupant passes through the door, the simulator calculates a delay time based on the specific flow of the door. That delay time is added to the door and must elapse before another occupant is allowed to pass through.

Each door may have a different specific flow depending on the direction occupants are going through the door and the type of terrain connected to the door. The specific flow for a particular direction through a door is

$$F_s = (1 - .266 * D) * k * D$$

The evacuation speed constant, k, depends on the terrain of the previous room, and D is the maximum of the occupant densities in the rooms attached to the door. Because the flow equation is quadratic, the value of D is clamped to the range $[1.9, 3.0]\ pers/m^2$. This range ensures that low densities do not slow the flow rate and that high densities do not reduce the flow rate to zero.

In the user interface, if *Door Flow Rates at High Density, Use a Calculated Specific Flow* is selected, the density is calculated as described above. Otherwise, it is set to $1.88\ pers/m^2$ to maximize the flow rate.

The time it takes n occupants to pass through a door with effective width W_e is:

$$T = \frac{n-1}{F_s}$$

The n value is reduced by 1 because the first occupant through a door does not have to wait for a time delay.

In counter-flow situations, an occupant from R_1 may be waiting at a queue to enter R_2 while an occupant from R_2 may be waiting to enter R_1. In this case, the queues evenly exchange their next occupant and both occupants are allowed through the door. The delay time placed on the door queue becomes the sum of the delay times from each occupant's passage, which maintains the correct flow rate for the simulation.

5.4. Collision Handling/Response

In SFPE mode, while occupants cannot collide with other occupants, they can still collide with walls. Collision handling is applied in two steps. The first step occurs before movement is attempted for a time step, and the second occurs during movement. For the pre-movement step, the travel velocity is adjusted to force the occupant to slide along any nearby walls. After the travel velocity is adjusted, the occupant attempts to move using this new velocity. During the movement stage, wall collisions are still possible, so the occupant will simply halt at the earliest collision.

Chapter 6. Steering Mode

In steering mode, Pathfinder uses a combination of steering mechanisms and collision handling to control how the occupant follows their seek curve. These mechanisms allow the occupant to deviate from the path while still heading in the correct direction toward their goal.

6.1. Velocity

As an occupant moves along their path, they calculate a modified maximum velocity, $\acute{v}_{max}$, that depends on the occupant's current terrain, specified maximum velocity, $v_{\max}$, and the spacing of surrounding occupants. The spacing of surrounding occupants is used to estimate an occupant density, D, as described below. These parameters are then used in the equations to calculate v in SFPE mode which becomes $\acute{v}_{\max}$.

In steering mode, both $v_f(D)$ and v_{ft}, which are used to calculate $\acute{v}_{max}$, may either be left at the SFPE default or may be user-defined in the occupant profile as piece-wise linear functions. $v_f(D)$ is entered as a function of occupant density and v_{ft} is entered as a function of either stair step slope or ramp slope, depending on the terrain type. Stair step slope is entered in the user interface by specifying a stair's *riser* and *tread*. Ramp slope is determined by the normal of the triangle that belongs to a ramp node and cannot be entered directly by the user.

A triangle's slope is calculated as follows:

$$slope = \frac{\sqrt{n_x{}^2 + n_y{}^2}}{n_z}$$

n_x, n_y, and n_z are the components of the triangle's normal. In addition, different $v_f(D)$ and v_{ft} functions may be defined for when the occupant goes up or down stairs or ramps. This is in contrast to SFPE calculations, which use the same functions for both up and down.

Once $\acute{v}_{max}$ is calculated, it is then used by the steering system to calculate a desired velocity vector as described in Section 6.4.

6.2. Acceleration

An occupant's acceleration is split into multiple components depending on the desired velocity vector calculated by the steering system.

A tangential forward component of acceleration is calculated as:

$$a_{fmax} = \frac{v_{max}}{t_{accel}}$$

Where t_{accel} is the occupant profile parameter, *Acceleration Time*.

The tangential reverse component of acceleration is:

$$a_{bmax} = 2 * a_{fmax}$$

The radial component of acceleration is:

$$a_{rmax} = 1.5 * a_{fmax}$$

These components are combined to determine the final acceleration vector.

6.3. Estimation of Occupant Density

To calculate $\acute{v}_{max}$ for an occupant, the occupant density D at that occupant's location must be known.

Pathfinder estimates the density by using the spacing of the near occupants and the average longitudinal and lateral spacing density relationship demonstrated in Chapter 3 of (Fruin and Strakosch 1987) as shown in Figure 13.

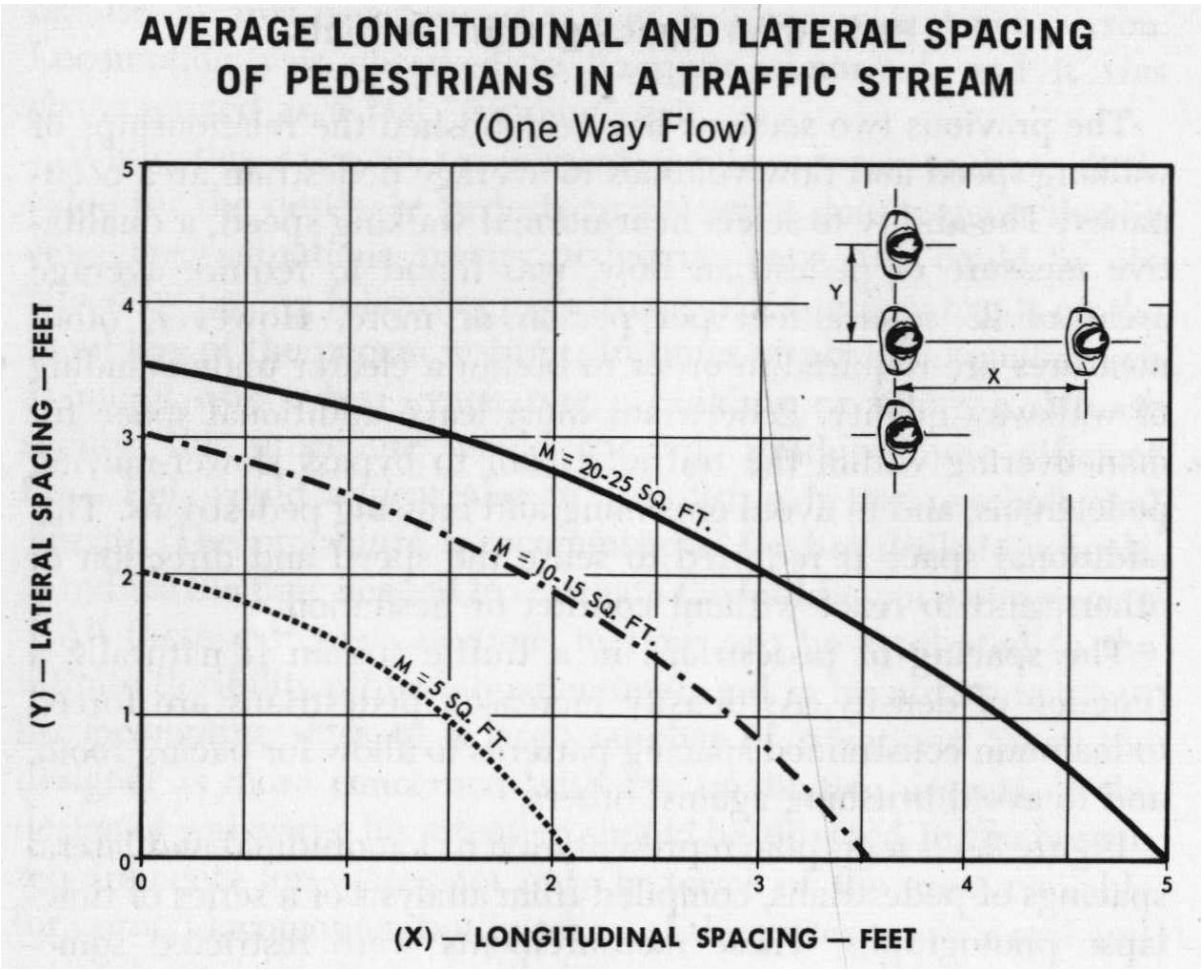

Figure 13. Average Longitudinal and Lateral Spacing of Pedestrians in a Traffic System (Figure 3.4 of Fruin's *pedestrian planning and design*)

In Pathfinder, the density lines in the figure are treated as contours, each being estimated as an ellipse. The contours are mirrored about `Y=0`.

To calculate the density for an occupant, the X axis in the figure is aligned with the occupant's current velocity and the origin is set to the occupant's location, forming a local coordinate system. Then for each other near occupant, their location is transformed to this local coordinate system. If the local coordinate for the other occupant has an x value less than 0, the occupant is ignored. This prevents an occupant's speed from being affected by occupants behind them. For occupants with local $x >= 0$, the density is interpolated or extrapolated from the density contours. The maximum of these densities is then used as the density for the occupant.

6.4. Steering

The steering system in Pathfinder moves occupants so they roughly follow their current seek curve and can respond to a changing environment. Inverse steering, used in Pathfinder, is the

process of evaluating a set of discrete movement directions for an occupant and choosing the direction that minimizes a cost function. See Figure 14 for an example of sample directions. The cost function is evaluated by combining several types of steering behaviors to produce a cost. The types of steering behaviors used are determined by the occupant's current state, and the number of sample directions is controlled by the occupant's state and current velocity. For more information on states, see Section 6.6.

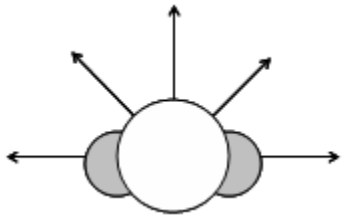

Figure 14. Sample inverse steering directions

Pathfinder defines several steering behaviors: *seek*, *idle separate*, *seek separate, seek wall separate, avoid walls*, *avoid occupants*, pass, *lanes*, and *cornering*. Most behaviors award a cost between 0 and 1 for each sample direction. The net cost for a direction is a weighted sum of these values.

6.4.1. Seek

The *seek* behavior steers the occupant to travel along a seek curve. Given the sample direction v, and a seek curve sc, the seek behavior bases its cost on the magnitude of the angle between v and the tangent to sc.

The cost is calculated as follows:

$$C_{seek} = \frac{\theta_t}{2\pi}$$

Where θ_t is the angle between v and the tangent vector to sc.

6.4.2. Idle Separate

The *idle separate* behavior steers occupants to maintain a desired distance away from other occupants and is used when occupants are in an *idle* state. This behavior works somewhat outside the inverse steering system, in that before considering sample directions, the separation behavior calculates a desired absolute movement vector (direction and distance).

This movement vector is calculated as the average of occupant separation vectors as follows:

$$\overline{m} = \frac{1}{n_{occ}} \sum_{i=1}^{n_{occ}} \overline{m_i}$$

Where n_{occ} is the number of occupants by which the occupant would like to separate.

If the i^{th} occupant is idle, $\overline{m_i}$ is calculated as:

$$D_{\text{gap}} = |\overline{p} - \overline{p_i}| - r - r_i$$

$$\overline{m_i} = (D_{\text{gap}} - D_{sep})\frac{\overline{p} - \overline{p_i}}{|\overline{p} - \overline{p_i}|}$$

Where $\overline{p}$ is the position of an occupant, r is the radius of an occupant, and D_{sep} is the desired separation distance of the occupant (settable via the input parameter, *Comfort Distance*).

If the i^{th} occupant is seeking, $\overline{m_i}$ is instead calculated such that it is perpendicular to the i^{th} occupant's direction of travel and its magnitude is defined as:

$$|\overline{m_i}| = r + r_i + D_{sep} - D_{path}$$

Where D_{path} is the occupant's distance to the nearest point on the line tangent to the i^{th} occupant's seek curve.

Once the movement vector is defined, the *Separation* behavior works like other inverse steering behaviors.

The cost is calculated as follows:

$$C_{isep} = 1 - (\overline{m_i} \cdot \overline{d_s})$$

Where $\overline{d_s}$ is the sample direction.

6.4.3. Avoid Walls

The *avoid walls* behavior detects walls and steers the occupant to avoid collisions with them. This behavior projects a moving cylinder ahead of the occupant in the direction of the projected point. The cost reported by this behavior is based on the distance the occupant can travel in the direction of the projected point. It is also affected by the angle at which the occupant hits the wall. The cost is decreased if the agent will hit the wall at a shallow angle to the desired direction.

$$D_{min} = \frac{\acute{v}_{curr}{}^2}{2a_{bmax}}$$

$$D_{max} = D_{min} + \left[\frac{\acute{v}_{max}{}^2}{2a_{bmax}}, \acute{v}_{curr} t_{wcr}\right]$$

$$C = 1 - \frac{D_{coll} - D_{min}}{D_{max} - D_{min}}$$

$$C_{aw} = \begin{cases} 1, & \overline{d_{slide}} \cdot \overline{d_{des}} \le 0 \\ C * (1 - \overline{d_{slide}} \cdot \overline{d_s}), & \overline{d_{slide}} \cdot \overline{d_{des}} > 0 \end{cases}$$

Where:

t_{wcr} is the maximum time at which an occupant will react to a wall collision (fixed at 2 s).

a_{bmax} is the maximum tangential deceleration

D_{coll} is the collision distance

$\overline{d_{slide}}$ is the direction the agent would slide if they hit the wall

$\overline{d_{des}}$ is the desired travel direction

$\overline{d_s}$ is the sample direction.

The resulting cost is clamped from 0 to 1.

6.4.4. Avoid Occupants

The *avoid occupants* behavior steers an occupant to avoid collisions with other occupants. This behavior first creates a list of occupants within a frustum whose size and shape is controlled by the velocity of the occupant. Then the behavior projects a moving cylinder ahead of the occupant in the sample direction. This cylinder is tested against another moving cylinder for each nearby occupant. If none of the moving cylinders collide the cost is zero, otherwise the cost is based on how far the occupant can travel prior to the collision. The closer this collision point, the higher the cost of the steering behavior.

The cost is based on the earliest collision with another occupant along the sample direction and is evaluated as follows:

$$D_{min} = D_{sep} + \frac{\acute{v}_{curr}{}^2}{2a_{max}}$$

$$D_{max} = D_{min} + \max\left[\frac{\acute{v}_{max}{}^2}{2a_{max}}, \acute{v}_{curr} t_{cr}\right]$$

$$C_{ao} = 1 - \frac{D_{coll} - D_{min}}{D_{max} - D_{min}}$$

Where:

D_{sep} is the desired separation distance between the occupant and the collision occupant (settable via the input parameter, *comfortDist*)

t_{cr} is the maximum time at which an occupant will react to a collision (settable via the input parameter, *collisionResponseTime*)

D_{coll} is the collision distance.

The resulting cost is clamped from 0 to 1.

6.4.5. Seek Separate

The *seek separate* behavior spreads out occupants to maximize their travel speed as calculated by the occupant's speed-density curve and Fruin's spacing-density relationship (see Figure 13).

For a sample direction, the occupant's future location along that direction is predicted using $\acute{v}_{max}$ and the *steering update interval*, settable in the user interface. In addition, the locations of the surrounding occupants are predicted using their current velocity and the *steering update interval*. From these predicted locations, the density D is estimated as described in Section 6.3. The speed is then predicted at that location from the density and the occupant's speed-density curve. The predicted speed is then used to calculate the cost.

$$C_{ssep} = 1 - \left(\frac{v_{pred}}{v_{max}}\right)^2$$

Where:

v_{max} is the occupant's maximum speed ignoring occupant density.

v_{pred} is the predicted speed.

6.4.6. Seek Wall Separate

The *seek wall separate* behavior steers occupants such that they want to main a *boundary layer* distance away from walls. Like the *seek separate* behavior, the occupant's location is predicted along the sample direction using $\acute{v}_{max}$ and the *steering update interval*. The nearest wall to this location is then used calculate the cost.

$$C_{swsep} = 1 - \frac{d_w - r - bl}{bl}$$

Where:

d_w is the distance to the nearest wall (walls more than 90° from the sample direction are ignored)

r is the occupant's radius

bl is the *boundary layer* set in the occupant profile. The cost is clamped from 0 to 1.25.

6.4.7. Lanes

The *lanes* behavior steers occupants into lanes when they detect that they are in counterflow with other occupants. It works by steering an occupant towards the center of mass of the occupants in front who are not in counterflow. Other occupants are considered to be in front if their center is within 60° of the tangent to the occupant's seek curve. For these in-front occupants, the vector to their center of mass is calculated as the following:

$$\overline{v_{cen}} = -\overline{p_{occ}} + \frac{1}{n_{occ}} \sum_{i=1}^{n_{occ}} \overline{p_i}$$

Where:

$\overline{p_{occ}}$ is the location of the occupant

n is the number of occupants in front

$\overline{p_i}$ is the location of the i^{th} occupant in front.

The cost, C_{lanes}, for the *lanes* behavior is determined in the following order:

1. If the occupant is considered to be a lane leader, the cost is 0. An occupant is a lane leader if there are no occupants in front who are not in counterflow.
2. If a test direction does not lead to counterflow, the cost is 0. A test direction leads to counterflow if at least one vector from the occupant to a counterflow occupant has an angle less than 36° to the test direction.
3. The cost is calculated as the angle between the test direction and $\overline{v_{cen}}$.

6.4.8. Pass

The *pass* behavior steers occupants so that they prefer to walk behind faster moving occupants. For each sample direction, the occupant calculates a collision with other occupants in that direction, treating the other occupants as stationary. If a collision is detected, the cost is calculated using the nearest collision as follows:

$$C_{pass} = 1 - \frac{v_o}{v_{max}}$$

Where:

v_o is the speed of the other occupant

v_{max} is the occupant's maximum speed, ignoring other occupants.

The cost is clamped to the range `[0-1]`. The pass behavior is not used if counterflow is detected.

6.4.9. Cornering

The *cornering* behavior seeks to steer agents so that they can take wide turns as part of a group without cutting in front of each other. This allows them to better utilize wide hallways/ramps with turns. To a certain extent, the *avoid occupants* and *seek separate* behaviors already achieve this, but the *cornering* behavior improves this.

The *cornering* behavior works similarly to *avoid occupants*, but it treats the size and positions of

the nearby occupants differently when calculating intersections between occupant paths. The size of a nearby occupant is expanded by 50% and their position is moved by a distance of 150% of the occupant's radius along their most recent steering direction.

In addition, a flow direction is calculated from nearby occupants who are in front of the occupant as follows:

$$\overline{v_{flow}} = \sum_{i=1}^{n} \overline{d_{if}}$$

Where:

n is the number of nearby occupants in front

$\overline{d_{if}}$ is the direction that the i^{th} occupant is facing.

The cost, C_{cnr}, is calculated as follows:

1. If any intersections are found, the cost is calculated the same as in *avoid occupants* except with the expanded occupant radius and adjusted positions.
2. If the test direction's angle with the tangent to the seek curve is greater than 60° and the test direction's angle with $\overline{v_{flow}}$ is greater than 90°, the cost is 1.0.
3. Otherwise, the cost is 0.

6.4.10. Final Direction Cost

The final cost for a sample direction is a weighted sum of the individual behavioral costs:

$$C_{ds} = .5C_{seek} + w_{isep}C_{isep} + w_{ao}C_{ao} + w_{aw}C_{aw} + w_{ssep}C_{ssep} + w_{swsep}C_{swsep} + w_{lanes}C_{lanes} + w_{cnr}C_{cnr} + w_{pass}C_{pass}S$$

The weights depend on the occupant's current state, and are defined in the following table.

Table 2. Weights by State

Weight	State=Idle	State=Seeking
w_{ao}	1	1
w_{aw}	1	1
w_{isep}	1	0
w_{ssep}	0	2
w_{swsep}	0	1
w_{lanes}	0	1
w_{cnr}	.2	.2

Weight	State=Idle	State=Seeking
w_{pass}	0	.5

6.5. Evaluating Movement

Once the lowest cost direction has been determined, the steering velocity and acceleration are calculated that will move the occupant in the steering direction.

Along with a cost, each steering behavior calculates a maximum distance that should be traveled along the sample direction. This maximum distance is then used to determine the magnitude of the desired velocity, $\overline{v}_{des}$, as follows:

$$D_{stop} = \frac{\dot{v}_{curr}{}^2}{2a_{max}}$$

$$|\overline{v}_{des}| = \begin{cases} 0, & D_{max} \leq D_{stop} \\ v_{max}, & D_{max} > D_{stop} \end{cases}$$

$$\overline{v}_{des} = |\overline{v}_{des}|\overline{d}_{des}$$

Where:

D_{max} is the maximum distance for the lowest cost sample direction, $\overline{d}_{des}$ is the lowest cost sample direction, and $\overline{v}_{curr}$ is the occupant's current velocity.

The acceleration is calculated as follows (Reinolds 1999):

$$\overline{a} = \frac{\overline{v}_{des} - \overline{v}_{curr}}{|\overline{v}_{des} - \overline{v}_{curr}|} a_{max}$$

Explicit Euler integration is then used to calculate the velocity and position of each occupant for the next time step from their steering acceleration. The velocity and position are calculated as follows:

$$\overline{v}_{next} = \overline{v}_{curr} + \overline{a}\,\delta t \quad \overline{p}_{next} = \overline{p}_{curr} + \overline{v}_{next}\delta t$$

Where:

δt is the time step size, $\overline{p}_{curr}$ is the current position, and $\overline{p}_{next}$ is the position after the time step.

6.6. Occupant States

Depending on an occupant's current scripted behavior, they will be in one of two states:

Seeking

The occupant is trying to follow a path to some destination.

Idling

The occupant is waiting for a specified amount of time.

6.6.1. Effect on Steering Behavior

Occupant state has a direct effect on which steering behaviors are combined to determine the lowest cost steering direction.

When idling, the occupant combines *separation*, *avoid occupants*, and *avoid walls*. This allows the occupant to maintain a separation with other occupants, move away from others that might be trying to seek near them, and avoid other occupants and walls at the same time.

When seeking, the occupant combines *seek*, *avoid occupants*, *avoid walls, seek separation, seek wall separation*, *lanes*, and *cornering*. This allows the occupant to avoid collisions with other occupants and walls and follow their seek curve. To a limited extent, the avoid occupants behavior integrates separation, so a separate separation behavior is unnecessary. While seeking, the occupant may temporarily switch to an idle state if they sense another occupant of higher priority or they have moved in such a way that they are touching another occupant. By temporarily switching to the idle state, they are able to move away from the other occupant to maintain the desired separation.

6.6.2. Effect on Sample Directions

Occupant state also affects the number of sample directions used for inverse steering.

In an idle state, 8 sample directions are tested, 45° apart, along with a "null" direction that tests standing still. This allows the occupant 360° of movement so they can easily separate from others.

When seeking, the occupant tests a different set of directions depending on the occupant's speed. A seed direction tangent to the occupant's seek curve is used as the starting direction. If the occupant's speed is relatively slow, they will consider 7 more sample directions, 45° apart as when idling. If their speed is faster, the sample directions are taken 15° apart, up to 75° to either side, creating 9 sample directions. The occupant's speed is considered "slow" if the following is true:

$$v_{curr} \leq f_{slow} \acute{v}_{max}$$

Where f_{slow} is settable via the input parameter, slowFactor, and $\acute{v}_{max}$ is the occupant's modified maximum velocity. In addition to the sample directions, the null direction is again considered for seeking.

6.7. Priority

Pathfinder provides a priority system that operates on discrete priority levels assigned to each occupant. When occupants encounter other occupants at the same priority level as their own, they behave as indicated above (the common case). If, however, they detect another occupant with a different priority nearby and in front of them, they will slightly alter the above behaviors.

If the other occupant is of lower priority, the occupant will not separate and will use a comfort distance of zero, effectively allowing them to push against the other occupant if necessary. Because there is no notion of occupants exerting force on one-another, the other occupant must respond accordingly.

So in the inverse case of an occupant detecting another of higher priority within their comfort distance, they will ignore their seek behavior and instead use their separation behavior, even if their goal puts them in a seek state. This allows them to back away from the occupant of higher priority, giving the higher priority occupant a chance to move through.

Priority levels are completely relative. For example, if three occupants meet having priorities of 5, 7, and 12, their behavior toward one another would be exactly the same as if their priorities were 0, 1, and 2, respectively. Occupants with higher priority values have higher priority over other occupants.

6.8. Resolving Movement Conflicts

There are some scenarios where an occupant's movement conflicts with another occupant's movement due to limitations of the geometry. In these situations occupants must negotiate how to resolve these conflicts such that they can continue moving.

The following examples illustrate how these situations can arise:

Figure 15 - Multiple occupants are simultaneously headed in a common direction and are approaching a physically tight area that will not allow all of them to pass at once (such as a narrow door or narrow hallway).

Figure 16 - Occupants are headed in opposite directions in a crowded hallway (counterflow).

Figure 17 - Occupants have squeezed into a tight area and cannot back up, due to a wall or one-way door.

Figure 18 - Occupants are headed in opposite directions in a hallway that will not physically allow them to pass.

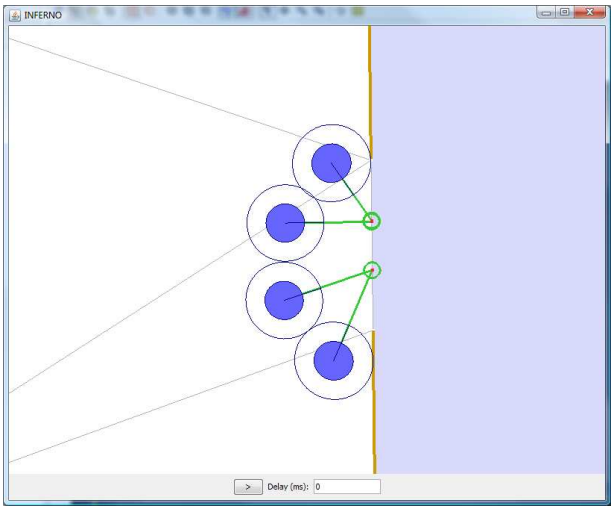

Figure 15. Occupants are headed toward a similar, conflicting waypoint.

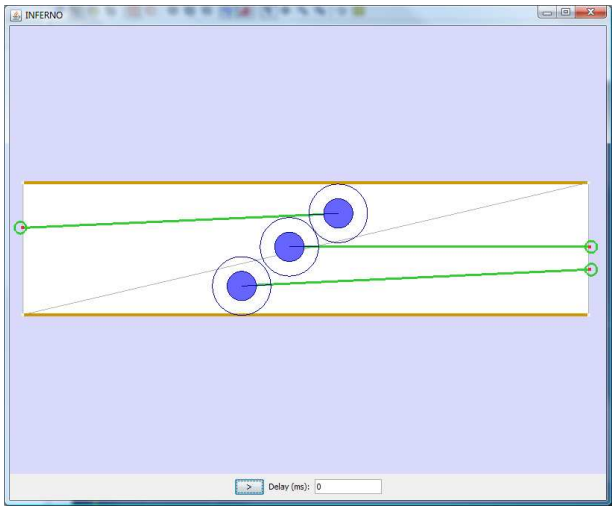

Figure 16. Occupants are headed toward opposing waypoints in a crowded hall.

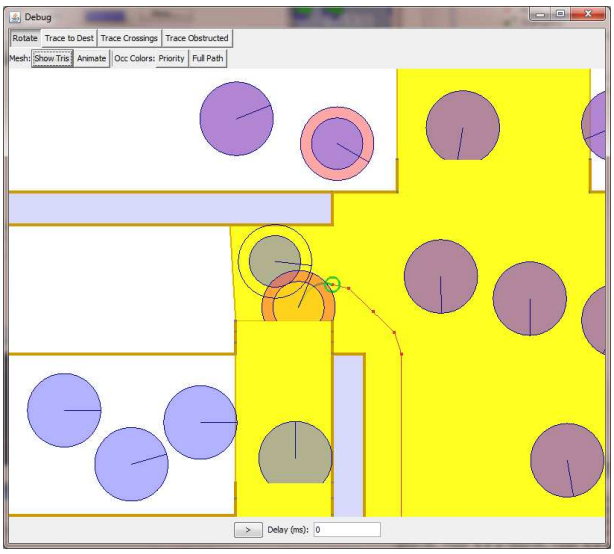

Figure 17. Merging occupants squeezed together

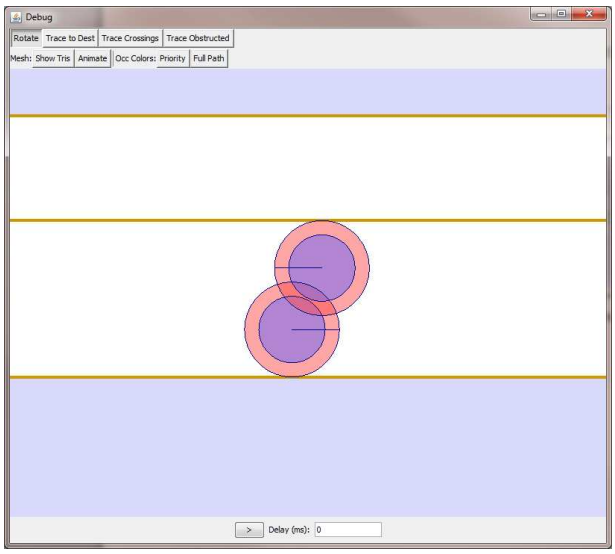

Figure 18. Counterflow occupants squeezed together

Pathfinder employs special handling to resolve these movement conflicts and prevent occupants from becoming stuck. This is handled as part of each occupant's steering behavior as follows:

1. The occupant performs their steering behavior as described previously and determines that the lowest cost direction is to either stand still or to move counter to their desired steering direction because of another occupant.
2. The occupant performs a "free pass" test as discussed below. If the occupant obtains a free pass, they continue the next step. Otherwise, they return the no-progress steering direction.

3. The occupant recalculates steering with a *locally-elevated priority*. A locally-elevated priority is one that makes the occupant appear to have a higher priority to others within the same priority level, but to other occupants with higher priority levels, the occupant still appears to have lower priority.
4. If the occupant makes progress with the newly calculated steering direction, the occupant raises their priority to this new locally-elevated priority (if not already raised) and returns the new steering calculation. If the occupant does not make progress, however, they skip to the next step.

NOTE

When an occupant raises their priority, they also begin a count-down timer (settable via the input parameter, *Persist Time*) or extend a previous timer if priority was already raised. In addition, the occupant's radius is reduced by a reduction factor (settable via the input parameter, *Reduction Factor* in the pre-processor), allowing the occupant to squeeze past other occupants. If other occupants detect this occupant, they too will reduce their radii by their set reduction factors.

5. If the occupant has not yet set a timer, they will return the no-progress result. If they have set a timer, they skip to the next step.
6. If the occupant's timer has not yet expired from a previous steering calculation, the occupant remains stationary with a locally-elevated priority in hopes that other occupants will separate due to the elevated priority. If the timer has expired, however, they skip to the next step.
7. The occupant maintains raised-priority and enters a state in which they can pass through the other occupants that are immediately in their way.

6.8.1. Free Pass

In steering mode, an occupant obtains a free pass if at least one of the following conditions is true for all nearby occupants:

- The other occupant is a lower priority.
- The other occupant is the same priority and has a lower chance of reaching the intersection of their paths before the occupant being considered can.

6.9. Collision Avoidance/Response

While the wall and occupant avoidance behaviors will attempt to steer around obstacles, they might not always succeed. This often occurs in crowded situations when occupants cannot avoid being pressed tightly against walls and other occupants. In these situations, additional collision handling is necessary to prevent the simulation from entering an invalid state. There are two

collision handling scenarios: one in which two or more occupants collide and another where an occupant collides with the boundary of the navigation mesh (i.e. a wall).

If collision handling is turned on, the occupant will halt at the earliest collision with either a wall or another occupant for a given time step. If collision handling is off, the occupant will only halt at the earliest collision with a wall.

6.10. Movement through Doors

By default, Steering Mode simulations provide no additional constraints on occupants when they move through doors. Flow limiting can be turned on, however, either through the simulation parameter, *Limit Door Flow Rate*, or the *Flow rate* parameter on individual doors.

In Steering mode, flow limiting works similarly to that in SFPE mode. For more details, see Section 5.3. The main differences between flow limits in the two modes are as follows:

- In steering mode, the flow limit can only be specified as a fixed value. It cannot be based on room densities as in SFPE mode.
- The actual achieved flow rate in steering mode will often be less than the specified limit. This is due to the acceleration model and occupant avoidance used in steering mode. When an occupant is stopped at a door, they have to accelerate again to leave the doorway and allow another occupant to enter.
- Occupants passing through a flow-limited door in steering mode may encounter a slight slow-down at the door threshold even if they do not need to be held at the door to achieve the flow limit. This is because each occupant always needs to be enqueued at the door when they cross the threshold in order for the limiting logic to progress properly. This enqueuing step stops the occupant completely. In situations such as these, the door will immediately release the occupant, but some of the occupant's momentum will be lost. This slowdown effect is dependent on the simulation time-step size. It will be worse for large time steps.

Chapter 7. Vehicle Agents

Occupants can be assigned body shapes other than the default cylinders. Any convex polygon can be used as occupants' body shape. Collision detection and avoidance of polygonal shapes is more complex than when only considering circular shapes. In this section we call occupants with polygonal shapes vehicle agents or vehicles.

Movement of vehicles is different from the movement of occupants. Vehicles are not allowed to move sideways, their rotation is restricted, and they follow a different curve.

Pathfinder is using Minkowski difference to perform collision avoidance between vehicles and walls as well as between multiple vehicles and combinations of vehicles and occupants with circular shapes.

For collision avoidance it is essential to be able to determine a distance to collision of two shapes. For polygonal shapes this distance is computed using Minkowski difference. The Minkowski difference consists of the convex hull of every point on the boundary of one shape subtracted from every point on the boundary of the other shape. It can be viewed as a process of shrinking one shape while simultaneously expanding the other. A distance from the origin to the closest point of the resulting shape is the collision distance. This method finds a collision distance of translating shapes. Since vehicles can also rotate around their origins, rotation of the shapes has to be also taken into account. Rotation is not included in the computation of the collision distance. However, collision detection (determining whether a vehicle can rotate by a certain degree without colliding) includes rotation in its movement calculations. More information about this calculation is available in "Collision detection for dummies" (Firth 2011).

In general, polygonal shapes can often appear in a state that is hard or impossible to navigate from, for example when a rectangular vehicle is touching a wall with one of its edges or when it is navigating through narrow doors or corridors. Because a developmental focus of Pathfinder is to never become stuck, vehicles are allowed to partially overlap with obstacles in difficult situations.

Chapter 8. Assisted Evacuation

In an assisted evacuation scenario, assistants help clients navigate to various targets. In order to receive assistance, clients must be using a vehicle shape that has at least one attached occupant position. Assistants repeatedly select clients, help them achieve their goals, and come back to assist more clients. When requesting assistance, clients are unable to move until assistants are attached to them in all attached occupant positions of the vehicle.

In Pathfinder, assisted evacuation is implemented using agents' *goals* (see Chapter 3 for more information about goals). The following sections describe the goals assigned to agents with different roles in the assisted evacuation scenario.

8.1. Assist Occupants Goal

This goal is specified as the **Assist Occupants** behavior action in the user interface. This goal allows an assistant to help all clients who are requesting or will request assistance from a particular team until all those clients have either exhausted their goals or detached from their assistants.

The following is a broad step-by-step procedure of how the assist occupants goal progresses:

1. An occupant begins the *Assist Occupants Goal* and joins an assisted evacuation team.
2. The assistant checks whether there are any current occupants or future occupants from an occupant source who will ever become clients of this team. If this condition holds true, the assistant continues to step 3. Otherwise, the assistant skips the following steps, leaves the team, and continues with their next goal.
3. The assistant enters an idle state and waits for clients to request assistance from the team.
4. The assistant chooses a single client who can be reached from the assistant's current location and meets one of the following criteria:
    a. The client has more assistants attached than any others.
    b. No other client meets criterion *a*, the client is in the team's priority list as defined in the user interface, and the client has the highest priority.
    c. No other client meets criteria *a* or *b*, and the client is closer than any others.
5. The assistant offers to assist the selected client. If the client rejects the offer, the assistant repeats steps 4 and 5. The client may reject if they have received more offers at once than they had remaining attached positions.
6. The assistant heads toward the chosen client. Once the client is reached, the assistant attaches to the client in one of the attached occupant positions of the client's vehicle.

7. The assistant becomes passive and waits for the client to detach from them.
8. The assistant repeats steps 2 through 8.

For each assistant, the *Assist Occupants Goal* is further decomposed into three other sub-goals:

Idle

Keeps the assistant in an idle state until a client has been chosen

Attach to agent goal

Engaged once a client has been chosen. The assistant navigates toward an assigned *attached occupant position* of the client's vehicle. When the assistant reaches this position, they become passive.

Passive mode goal

Once the assistant is attached to the client, they turn passive. This means that their movement will depend on the movement of the client. The client will give the assistant an updated position to navigate toward and the assistant only needs to modify its velocity vector in a way that moves it to this given position. The assistant stays in this state until the client detaches.

The process of assigning these goals in a multithreaded application can lead to problems such as the loss of determinism, concurrency exceptions or increased running time due to the necessary synchronization. All these issues are addressed by the coalition formation algorithm described in Section 8.3.

8.2. Wait for Assistance Goal

This goal is specified as the **Wait for Assistance** behavior action in the user interface. The procedure for this goal is as follows:

1. An occupant starts a *Wait for Assistance* goal, becoming a *client* of a set of teams.
2. The client waits indefinitely for an assistant from one of the teams to offer assistance.
3. From all of the occupants who have offered assistance, the client chooses the closest ones and assigns them to the remaining empty attached occupant positions. If there are any extra offers, they are rejected.
4. The client repeats steps 2 and 3 until all attached occupant positions are filled.
5. The client expands their vehicle shape to include the locations of the attached assistants. This change is not visible in the results view, but can be observed in the *debug simulation* view
6. The client ends the *Wait for Assistance* goal.

At this point, the client has attached assistants. The client then continues with their next goals as

they normally would. They detach from their assistants once one of the following occurs:

- If the client encounters a *Goto Exits* goal, they detach from all assistants shortly before going through the exit and then proceed to go through the exit on their own, even if they cannot move without assistance. They are then removed from the simulation.
- If the client encounters a *Detach from Assistants* goal, the client detaches from all assistants and continues on to the next goal on their own.
- If the client accomplishes all goals, they detach from all assistants and are removed from the simulation.

NOTE In the user interface, an occupant may be assigned a **Goto Refuge Rooms** behavior action. There is no corresponding goal in the simulator.

Instead, a client is assigned the following sequence of goals:

1. *Goto Rooms* – this tells the client to go to the refuge rooms.
2. *Fill Room* – this indicates that the client should move away from incoming doors and continue moving until an obstacle is encountered. This allows more occupants to fill the room.
3. *Detach from Assistants* – this allows assistants to detach and continue helping other clients.

8.3. Coalition Formation Algorithm

Pathfinder is a multithreaded application, in which the agents decide about their movement in a pre-move step and update their state in an update step. Each one of these steps can be performed by the agents in parallel. Assisted evacuation requires the agents to coordinate and cooperate. This behavior brings multiple implementation challenges, because it requires deterministic interactions between agents.

The following fundamental steps are performed by the simulator engine in every step of the simulation:

- *Pre-move agents and goals*
- *Update agents*
- *Update goals*

Notice that the goals are updated in a similar way to the agents. These three steps must be separated in order to preserve determinism.

The following are the procedures added to Pathfinder to support assisted evacuation:

Pre-move agent

The behavior of assistants changes in this step only if they are in the passive mode, in which case their movement calculation is based on the position required by the clients. The behavior of clients does not change.

Pre-move goals

In this step the agents are allowed to act on each other. If an assistant decides to assist a client, they make a reservation with that client. Similarly, if they decide to stop assisting the client, the assistant cancels their reservation. When a client receives a reservation, they store it in a list of reservations. Note that the order of this list is not deterministic at this time. The assistants also re-compute their distances to all possible clients and possibly select a new goal.

Update agents

The clients compute new positions for the assistants. They also resolve the reservations in the following way:

1. The list of reservations is sorted based on the distances from the assistants to the client.
2. The first reservations in the sorted list are chosen for the open vehicle positions.
3. The reservation list is cleared.

Update goals

In this step the agents are allowed to read each other's states. The assistants check their reservations to find out whether they have been assigned a spot to attach to. If so and the client is fully reserved, the assistant sets the new goal towards the client. If the client is not fully reserved, the assistant has to decide whether to stay with this client or leave and go help other assistants. This decision making is done as follows:

1. The assistant finds the closest other partially occupied client.
2. The assistant compares the pre-calculated distance to this client with the minimum of pre-calculated distances of assistants that reserved a spot with the other client. The side that has the higher distance is allowed to stay, while the other side has to cancel its reservation and switch to the other client by making a reservation in the pre-move step. In this way the assistants together minimize the distance to the client they are going to assist.

   NOTE This calculation is fully distributed, since each agent runs the comparison and decides about its behavior separately.

3. If the assistant did not find any other partially occupied clients, they will search for clients that only require assistance of a single agent and reserve a spot with it (in the pre-move step). Also in this step the assistants can turn passive (when attached to a client). This

cannot be done in the pre-move step, because agents consider the passive state of other agents when deciding about their movement.

Chapter 9. Elevators

Elevators in Pathfinder are composed of a discharge node and any number of elevator levels. Rather than modeling elevators using dynamic geometric elements (i.e. a floor that moves up and down), Pathfinder represents elevators using ordinary rooms. here is a room at each level the elevator touches, with doors connecting the elevator rooms to each level. There is one discharge room at the discharge level. There are several pickup rooms, one at each pickup level. Occupants that have entered a pickup room are moved directly toward the discharge room, bypassing any geometry in-between. They still appear to move, however, as if they are in a moving room as they are moved toward the discharge room. Elevator rooms that are not currently being served by the elevator are blocked using closed doors.

Elevator operation can be split into three main stages:

1. Idling
2. Pickup
3. Discharge

9.1. Idling

At the start of a simulation, each elevator is idle. All doors connecting all levels to the elevator are closed, the elevator is at the discharge floor, and the elevator is waiting to be called from one of the pickup levels. Once called, it proceeds to the pickup stage.

9.1.1. Elevator Calling

In order for an occupant to call an elevator, they must be assigned an elevator goal. If the elevator goal specifies multiple elevators to choose from, the occupant uses a modified version of Locally Quickest to choose the best one. This version of Locally Quickest adds the elevator doors on the level closest to occupant’s location to the list of door targets. The global travel time for each elevator door reflects an estimate of the time it will take for the elevator to reach the pickup level for that door as well as the time to travel in the elevator to the discharge level and walk to the nearest exit.

Once the occupant has chosen one of the elevator doors, the occupant walks to the door. When they are within .5 m of the chosen elevator door and the elevator is in the idle state, the elevator is called. If the elevator is part of a call group, all elevators in that group are called.

9.2. Pickup

When a pickup level calls an elevator, the elevator moves toward that level. If multiple pickup

levels call the elevator at once, the elevator travels to the one with the highest priority as specified in the user interface. As the elevator travels toward the pickup level, it continues monitoring calls. If a call is received from a level with higher priority before the elevator reaches the current pickup level, the elevator switches pickup levels. The time to reach the pickup level is specified in the user interface.

When the elevator reaches a pickup level, it opens its doors and waits for a countdown timer to elapse. The elevator then closes its doors and proceeds to the discharge stage.

9.2.1. Countdown Timer

When the elevator doors open at a pickup level, the elevator begins a countdown timer, initialized to the time specified by the *Open Delay* input parameter. Once this timer reaches zero, the elevator doors close. While the timer is counting down and the elevator has room for more occupants, two events may occur that can affect the timer:

- An occupant enters the elevator.
- An occupant is outside the elevator desiring to enter it, is moving more than 10% of their maximum speed toward the elevator, and is within 7 m of one of the elevator doors.

If one of these events occurs, the countdown timer is reset to be the larger of its current value or the *Close Delay* input parameter.

9.2.2. Loading

The process of loading an elevator and achieving the user-specified nominal load is controlled by occupants and their movement behavior rather than elevator logic. Pathfinder provides different mechanisms for achieving the nominal load depending on the simulation mode.

In SFPE mode, achieving the nominal load is fairly trivial. Usually, a room is assigned a maximum occupant density from the simulation parameters. The doors leading to that room use a gating mechanism to prevent occupants from entering if doing so would exceed the maximum density in the room. For elevator rooms, the maximum occupant density is derived from the nominal load as nominal_load/area. Because occupants can overlap in SFPE mode, no additional work is needed to ensure that the nominal load can be met.

In steering mode, however, Pathfinder requires further action to ensure the nominal load. This is because each room has a physical loading limit due to room shape, occupant size, and avoidance behavior. To allow elevators to reach the nominal load, collisions are turned off, while still allowing occupants to avoid one-another.

9.3. Discharge

After the elevator is loaded at the pickup level, it moves toward the discharge level. The amount of time it takes to reach the discharge level is the same as travelling from the discharge to pickup level as specified in the user interface. Once the elevator reaches the discharge level, it opens its doors. When the elevator is empty, it closes its doors and enters the idle state again.

Chapter 10. Solution Procedure

Pathfinder runs in a simulation loop that calculates movement at discrete time steps. For each time step, the following steps are carried out:

1. Update each occupant's current target point. This step takes the longest on the first time step because each occupant must find a path to their goal.
2. Calculate each occupant's steering velocity. The steering velocity will be calculated differently depending on whether SFPE or Reactive Steering mode is active.
3. Increment the current time step.
4. Move each occupant. This involves several sub-steps:
    a. Calculate the velocity for the current time. If steering mode is on, this will calculate a desired steering force from the desired velocity, and then use integration to calculate the actual velocity. In SFPE mode, this will simply be set to the desired velocity.
    b. If collision avoidance is turned on, detect potential collisions, and modify the desired velocity to avoid the collisions.
    c. Integrate the final velocity to find the maximum travel distance, and travel along the mesh until this distance is reached or until the earliest collision.
5. Update output files.

Chapter 11. Pathfinder Input File Format

The simulator portion of Pathfinder can optionally use an input file to run simulations. By default, this input file is written every time a simulation takes place. This section describes the input file including its format and all parameters.

11.1. NODES

Rooms, doors, and stairways are represented by nodes. At any given time during the simulation each occupant is either inside one of the following nodes or has exited the simulation.

```
[nodes]
name <node display index>, <animation id>,
[optional: "count" <max occupant count in pers> or "dens" <max occupant density in
pers/m^2>]
```

Table 3. Nodes

name: string	Name of a node
node display index: int	Index of the node display properties
animation id: int	Index of the node animation
max occupant count in pers: int	Maximum number of occupants allowed in the node
max occupant density in pers/m^2 : float	Maximum occupant density allowed in the node

11.2. VERTS

This section contains all of the vertices that will be used by the geometry (triangles and edges) in the input file.

```
[verts]
x y z
...
```

Table 4. Verts

x : float	x-coordinate
y : float	y-coordinate

z : float	z-coordinate

11.3. NAVMESH << NODES, VERTS

This section defines the walkable space within a simulation.

```
[navmesh]
ixnode ttype ixverta ixvertb ixvertc
...
```

Table 5. Navmesh

ixnode : int	Index of the node associated with this triangle
ttype : string	Terrain type: [open, stair]
ixverta : int	Index of first vertex
ixvertb : int	Index of second vertex
ixvertc : int	Index of third vertex

NOTE The order of the three vertices is significant. Use CCW ordering (i.e. right hand rule) to define the top of the mesh element.

11.4. DOORS (optional) << NODES

This section defines the doors that will be used if the `use_door_queues` option is enabled. Associating a door entry with a node causes that node to be recognized by the simulator as a door node and prevents the density calculation from being used to control the speeds of occupants within triangles associated with that node. This section does not define the geometric edges that the door represents.

NOTE Agents will not enter a door's queue unless they cross a special door edge (defined in the EDGES section).

Exit doors should define only one adjoining room and internal doors should define two such rooms. These entries are used to prevent overcrowding as occupants transition between rooms and to provide for more elaborate merge calculations.

```
[doors]
ixnode eff_width ixnodeA ixnodeB flowrate doorDir
...
```

Table 6. Doors

ixnode : int	Index of the node corresponding to this door
eff_width : float	Effective width of this door
ixnodeA : int	Index of a room adjoining the door (use dash "-" for none)
ixnodeB : int	Index of a room adjoining the door (use dash "-" for none)
flowrate: float	Flowrate of the door
doorDir: string	Specify either "dir+" (for a positive direction), "dir-" (for a negative direction), or use dash "-" for multidirectional.

11.5. EDGES (optional)

The entries in this section represent the geometric portion of entities that are defined as edges in the NAVMESH.

```
[edges]
etype <depends on etype>
...
```

Table 7. Edge

etype : string	Edge type: [boundary, door, exit_door]

```
boundary ixverta ixvertb
```

This edge type represents a boundary that occupants will not walk across.

Table 8. Boundary

ixverta : int	Index of the first vertex
ixvertb : int	Index of the second vertex

```
door ixnode ixverta ixvertb
```

This edge type represents an internal door. Doors of this type will not be included in the search for the nearest exit and should have two adjoining nodes defined in the corresponding door record.

Table 9. Door

ixnode : int	Index of the node corresponding to this door
ixverta : int	Index of the first vertex
ixvertb : int	Index of the second vertex

```
exit_door ixnode ixverta ixvertb
```

This edge type represents an exit door. Doors of this type will be included in the search for the nearest exit and should have one adjoining node defined in the corresponding door record.

NOTE

Edge definitions must match edge definitions in the NAVMESH. If a door is formed by the three vertices *A*, *B*, and *C*, it must be given as two edges: A-B and B-C. Edges that do not correspond to edges in the NAVMESH (e.g. A-C) are invalid and may cause the simulation to crash or otherwise behave unexpectedly.

11.6. PARAM (optional)

This section allows you to customize global simulation parameters. The format is a list of key value pairs.

```
[param]
key value
...
```

Table 10. Param Structure

key : string	The name of a simulation parameter
value : mixed	The value for a simulation parameter (type depends on *key*)

Table 11. Params

max_time	0	Simulation time limit in seconds (0=infinite)
show_vis	0	Turn debugging visualization on/off (0=off, 1=on)
dt_init	0.025	Simulation time step size (s)
dt_vis	0.25	Frequency of visualization output (s)
dt_wall_meta	0.5	Frequency of simulation progress meta data (s)
dt_csv_data	1.0	CSV data print time increment (s)
dt_snapshot	120.0	
handle_collisions	1	Turn collision handling on/off (0=off, 1=on)
reactive_steering	1	Turn reactive steering on/off (0=off, 1=on)
inertia	1	Turn inertia on/off (0=off, 1=on)
vel_from_density	1	Turn density-based velocity calculation on/off (0=off, 1=on). Only applies when inertia is off.
density_max	3.55	Maximum room fill density. Only applies when door queues are active.
specific_flowrate_max	1.32	Maximum specific flowrate for doors.
door_flow_from_density	1	Turn door flowrates as a function of density on/off (0=off, 1=on)
door_flow_density_min	1.9	The lower bound on density for calculating specific flow through doors (pers / m^2)

door_flow_density_max	3.0	The upper bound on density for calculating specific flow through doors (pers / m^2)
boundary_layer	0.150	Boundary layer used for SFPE simulations (m)
min_flowrate_factor	0.1	Influences calculation of minimum door flowrates. Lower values improve occupant recognition of slow door queues, but can result in excessive queue switching.
out_summary	*summary.txt*	Summary of room and door clearing times.
out_performance	*performance.txt*	Summary of performance information for a completed simulation.
out_snapshot_base	*snapshot*	The base name for generated snapshots.
out_results	*results.pfr*	The 3D Results file.
out_occ_time_history	*vis.pfd*	Movie playback (time history) output.
out_geom_time_history	*geomvis.pfg*	Movie playback (time history) output for geometry.
out_room_usage	*rooms.csv*	Room usage over time.
out_door_usage	*doors.csv*	Door usage over time.

11.7. EVENTS

Event entries describe events that will happen during the simulation.

```
[events]
time event
...
```

Table 12. Events

_time_float	simulation time at which the event will occur

event	one from the following list

- *close_door <door index>*
- *open_door <door index>*
- *change_door_dir <x coordinate> <y coordinate> <door index>*
- *set_speed_mod <"CONSTANT" or "FACTOR"> <modifier value>*

11.8. BEHAVIORS

These entries are descriptions of artificial intelligence for simulation agents.

```
[behaviors]
key value
...
```

Behaviors are stored in a JSON key-value format. The supported keys are `name` and `script`. Script contains a list of actions separated by a semicolon. Following is a table of available actions for occupant behavior in Pathfinder.

Table 13. Behaviors

Action	Encoding	Example
Go to exit(s)	`goto exit` <*comma separated list of exit door indices*>	`goto exit 1, 2`
Go to any exit	`goto exit any`	
Go to waypoint	`goto point` (*x*, *y*, *z*) *radius*	`goto point (1.5, 2.5, 0.0) 1.5`
Go to room(s)	`goto room` <*comma separated list of room indices*>	`goto room 1, 3`
Go to elevator(s)	`goto elevator` <*comma separated list of elevator indices*>	`goto elevator 0, 1`
Go to any elevator	`goto elevator any`	
Wait	`wait curve` <*curve index*>	`wait curve 0`
Wait for assistance by assisted evacuation team(s)	`get_assistance` <*comma separated list of indices of assisted evacuation teams*>	`get_assistance 0, 1`

Action	Encoding	Example
Fill the occupant's current room by moving away from doors	`fill_room`	
Fill the occupant's current room if their previous goal indicated they should	`fill_room_if_necessary`	
Detach from assistants	`detach_assistants`	
Assist occupants	`assist` <*index of assisted evacuation team*>	`assist 0`
Add tag	`tag` (<*tag name*>)	`tag (last_goal_started)`
Remove tag	`untag` (<*tag name*>)	`untag (some_tag)`
Wait until all occupants are tagged	wait_until_all_tagged (<*comma separated list of tags*>)	`wait_until_all_tagged (+, finished)`
Set profile property	`set_prop` <*key*>, <*value*>	`set_prop OccProfile.REMOVE_WHEN_FINISHED, {{val{:[{{false{:1.0}]}"}`

Example of a behavior line:

```
2: {"name":"Behavior01","script":"goto point (-7.0, 3.0, 0.0) 1.5; tag
(last_goal_started); goto exit any"}
```

11.9. PROFILES

```
[profiles]
key value
...
```

This section lists all the properties defined for an occupant profile. Profiles are stored in a JSON key-value format. Following table shows available keys and values for a profile. Note that only properties with values that are not default are written in the file. Furthermore, only profiles that are used by any occupant or occupant source are written in the file.

Table 14. Profiles

Key: "OccProfile._"	Value description	Value example
ACCEL_TIME	*<index of acceleration time curve>*	0
BOUNDARY_LAYER	*<index of wall boundary layer curve>*	0
COLLISION_RESPONSE_TIME	*<index of collision response time curve>*	0
COLOR	*<color in RGBA 0-1 format>*	`[1.0,0.0,0.0,1.0]`
CURR_DOOR_PREF	*<index of current door preference curve>*	0
DIST_TRAVELLED_DOUBLE_DIST	*<index of current room distance penalty curve>*	0
DESC	*<String description of the profile>*	"My profile"
FUNDAMENTAL	*<index of speed-density profile function>*	0
INIT_ORIENT	*<index of initial orientation curve>*	0
LOCAL_QUEUE_TIME_FACTOR	*<index of current room queue time curve>*	0
LOCAL_TRAVEL_TIME_FACTOR	*<index of current room travel time curve>*	0
MAXVEL	*<index of speed curve>*	0
MIN_SQUEEZE_FACTOR_CONST	*<reduction factor between 0.0 and 1.0>*	0.7
NAME	*<name of the profile>*	"Default"
OBEY_ONEWAY_DOORS	*<whether one-way-doors should be obeyed>*	`[{"false":1.0}]`
OCCMODEL	*<list of occupant models with associated weights>*	`[{"md5 /AsWom0003 /AsWom0003. bea":0.5},{"md5 /BMan0001 /BMan0001. bea":0.5}]`
PERSIST_TIME	*<index of persist time curve>*	0
PRINT_EXTRA_OUTPUT	*<whether CSV data should be printed>*	`[{"true":1.0}]`

Key: "OccProfile._"	Value description	Value example
PRIORITY_LEVEL	*<priority level, 2 in following example>*	`[{"2":1.0}]`
FUNDAMENTAL_RAMP_DOWN	*<speed density ramp down function>*	`{"x":[0.55,3.23], "y":[0.93,0.15], "type":"pw", "extrapolate": false}`
FUNDAMENTAL_RAMP_UP	*<speed density ramp up function>*	`{"x":[0.55,3.23], "y":[0.93,0.15], "type":"pw", "extrapolate": false}`
SPEED_RAMP_DOWN	*<index of ramp down speed function>*	0
SPEED_RAMP_UP	*<index of ramp up speed function>*	0
REAC_TIME	*<index of reaction time curve>*	0
REMOVE_WHEN_FINISHED	*<whether occupants should be removed when finished>*	`[{"true":1.0}]`
REQUIRES_ASSISTANCE	*<whether occupants require assistance to move>*	`[{"true":1.0}]`
RESTRICTED_COMPONENTS	*<index of the component restrictions line in the [component-restrictions] section>*	0
SLOW_FACTOR	*<index of slow factor curve>*	0
PROP_SPACING	*<comfort distance curve>*	`{"val": {"min":"0.08 m", "max":"0.1 m", "type":"unif"}, "type": "COMFORT_DIST"}`

Key: "OccProfile._"	Value description	Value example
FUNDAMENTAL_STAIR_DOWN	*<speed density stair down function>*	`{"x":[0.55,3.23], "y":[0.93,0.15], "type":"pw", "extrapolate": false}`
FUNDAMENTAL_STAIR_UP	*<speed density stair up function>*	`{"x":[0.55,3.23], "y":[0.93,0.15], "type":"pw", "extrapolate": false}`
SHAPE	*<index of occupant shape>*	0
SPEED_STAIR_DOWN	*<index of stair down speed function>*	0
SPEED_STAIR_UP	*<index of stair up speed function>*	0
TAIL_TIME_FACTOR	*<index of global travel time curve>*	0
WALK_ON_ESCALATORS	*<whether occupants can walk on escalators>*	`[{"true":1.0}]`

Example of a profile line:

```
0: {"OccProfile.NAME":"Profile00","OccProfile.SHAPE":1,
"OccProfile.COLOR":[0.2,0.4,0.8,1.0]}
```

11.10. FUNCTIONS

```
[functions]
key value
...
```

Functions are used to map an independent variable x to a dependent variable y. Pathfinder supports two types of functions: piecewise linear and constant. Functions are stored in a JSON

key-value format. Following table shows available keys and values for the piecewise linear function.

Table 15. Piecewise linear function

Key	Value description	Value example
`"x"`	[*<comma separated list of x values>*]	`[0.1, 0.7, 1.5]`
`"y"`	[*<comma separated list of y values>*]	`[1.9, 0.65, 2.3]`
`"type"`	*<type of function,* "pw" *for piecewise linear>*	`"pw"`
`"extrapolate"`	*<true/false depending on whether points that are out of scope of x values should be extrapolated>*	`false`

Constant function stores a single value. Following table shows available keys and values for the constant function.

Table 16. Constant function

Key	Value description	Value example
`"val"`	*<constant value>*	`0.1`
`"type"`	*<type of function,* `"const"` *for constant>*	`"const"`

Example of a function line:

```
0: {"x":[0.55,3.2779887218045123],"y":[1.0,0.15],"type":"pw","extrapolate":false}
```

11.11. DISTRIBUTIONS

```
[distributions]
key value
...
```

Distributions in Pathfinder are either curves or urns. Curves represent continuous distributions, while urns represent discrete distributions.

Following four curve types are currently supported: constant curve, uniform curve, standard normal curve, and log normal curve. Curves are stored in a JSON key-value format. Following table shows available keys and values for the constant curve.

Table 17. Constant curve

Key	Value description	Value example
"val"	*<constant value>*	`"12.2 cm"`
"type"	*<type of curve,* `"cc"` *for constant>*	`"cc"`

Following table shows available keys and values for the uniform curve.

Table 18. Uniform curve

Key	Value description	Value example
"type"	*<type of curve,* "unif" *for uniform>*	`"unif"`
"min"	"*<minimum of the uniform interval>*"	`"5.5 cm"`
"max"	"*<maximum of the uniform interval>*"	`"6.5 cm"`

Following table shows available keys and values for the standard normal and log normal curves.

Table 19. Standard and log normal curve

Key	Value description	Value example
"type"	*<type of curve,* "stdNorm" *for standard normal,* "logNorm" *for log normal>*	`"stdNorm"`
"mean"	*<mean value>*	`"6 cm"`
"stDev"	*<standard deviation>*	`"0.5 cm"`
"min"	*<minimum of the uniform interval>*	`"5.5 cm"`
"max"	*<maximum of the uniform interval>*	`"6.5 cm"`

Example of a curve line:

```
2: {"min":"45.58 cm","max":"50.58 cm","type":"unif"}
```

Urns are also stored in a JSON key-value format. An urn line contains two keys: "urn" and "type". The "urn" key maps to an array of pairs, each pair contains a value mapped to its probability. The sum of all probabilities in an urn must be equal to 1.0. The "type" key maps to the type of variables in the urn. The following types are supported: "int" for integer values, "bool" for true/false values, "string" for string values, and "ud" for double values with units.

Example of an urn line:

```
3: {"urn":[{"2.0 s":0.33},{"8.0 s":0.33},{"15.0 s":0.33999999999999997}],"type":"ud"}
```

11.12. COMPONENT RESTRICTIONS

```
[component-restrictions]
key value
...
```

This section defines component restrictions for occupants and occupant profiles. Component restrictions are stored in a JSON key-value format. Following table shows available keys and values for the component restrictions.

Table 20. Component restrictions

Key	Value description	Value example
"components"	*<indices of components that the occupants can or cannot access>*	`[0,4,1]`
"action"	_<whether the components are accepted or rejected. Possible values are "accept" and "reject"> _	`"reject"`

11.13. OCCUPANT SHAPES

This section defines shapes of occupants, both for polygonal and cylindrical occupants.

```
[occshapes]
type height attachedAgentsPositions anim model
...
```

Table 21. Polygonal occupant

type	poly <number of points> <space separated x, y, z coordinates of points>
height	<height in meters>
attachedAgentsPositions	<number of attachment positions> <space separated x, y, z coordinates of attachment positions>
anim	<animation>
model	<3D model>

Example of a polygonal occupant shape:

```
0: type=poly 4 -0.66 -0.38 0.0 0.66 -0.38 0.0 0.66 0.38 0.0 -0.66 0.38 0.0, height=1,
attachedAgentsPositions=1 -0.71 0.0 0.0, anim=wheelchair,
model=props/Wheelchair/Wheelchair.bea
```

Table 22. Cylindrical occupant

type	cylinder
diameter	<diameter in meters>
height	<height in meters>
[optional]	
diameterdist	<index of diameter distribution>
heightdist	<index of height distribution>
geomdiameter	<geometry diameter in meters>

NOTE *diameter* and *height* can be specified either as constant values or distributions. Distributions are referenced by the order in which they are specified in the DISTRIBUTIONS section. The geomdiameter parameter is optional. If it is omitted, the resulting diameter is used as the geomdiameter.

Example of a cylindrical occupant shapes:

```
1: type=cylinder, diameterdist=10, height=1.8288, geomradius=0.33
2: type=cylinder, diameter=.42, heightdist=3
```

11.14. ASSISTED EVACUATION TEAMS

This section defines assisted evacuation teams.

```
[assisted-evac-teams]
goals name
...
```

Table 23. Evacuation Team

goals	<bar separated list of occupant ids *OR* `NA` if the order to assist is not specified>
name	<name of the assisted evacuation team>

Example of an assisted evacuation team line:

```
1: goals=3|1,name=AETeam00
```

11.15. OCCUPANT SOURCES

```
[occupant-sources]
key value
...
```

This section defines occupant sources. Occupant sources are stored in a JSON key-value format. Following table shows available keys and values for the occupant sources.

Table 24. Occupant Source

Key	Value description	Value example
"name"	*<name of the occupant source>*	"OccSource00"
"component"	*<index of a node to which the source is attached>*	2
"behaviors"	*<distribution of behaviors>*	*see the example below*
"profiles"	*<distribution of profiles>*	*see the example below*

Key	Value description	Value example
"bounds"	*<bounding box of the occupant source specified by min and max points>*	*see the example below*
"flowrate"	*<index of the flow rate function>*	3
"enforceFlowrate"	*<whether the flow rate should be enforced regardless of occupants in the way>*	false

Example of an occupant source line:

```
1: {"component":13,"behaviors":{"urn":[{"2":0.4},{"0":0.6}]},"enforceFlowrate":false,"nam
e":"OccSource01","profiles":{"urn":[{"0":0.25},{"1":0.25},{"2":0.25},{"3":0.25}]},"bounds
":{"min":"(-3.0686769149632847, 5.422381629126958, 0.0)","max":"(-2.2558769149632854,
5.422381629126958, 0.0)"},"flowrate":4}
```

11.16. TAGS

```
[tags]
key value
...
```

Tags sources are stored in a JSON key-value format. Following table shows available keys and values for the tags.

Table 25. Tags

Key	Value description	Value example
"name"	*<name of the tag>*	"last_goal_started"
"desc"	*<description of the tag>*	"Last goal started"
"predefined"	*<whether the tag is predefined. Predefined tags include:* finished, exited, report_refuge_reached>	false

Example of a tag line:

```
0: {"name":"last_goal_started","predefined":false,"desc":"Last goal started"}
```

11.17. OCCUPANTS

```
[occupants]
key value
...
```

This section describes the occupants present at the beginning of the simulation. The starting node for each occupant is inferred based on the location of the occupant. Occupants' properties are stored in a JSON key-value format. Only properties that differ from the properties of an occupant's profile are stored in the occupants section. The following table shows only properties that must be always present, since there are no matching profile properties (or for implementation reasons). All Profile properties can also be defined for individual occupants. However, curves are replaced with specific values, and units are omitted. It is assumed that these values are entered in SI units.

Table 26. Occupants

Key	Value description	Value example
"name"	*<name of the occupant>*	"00001"
"id"	*<id of the occupant>*	0
"behavior"	*<index of the behavior of the occupant>*	1
"loc"	*<initial location of the occupant>*	"-2.1 -0.3 0.0"
"OccProfile.REAC_TIME"	*<reaction time of the occupant>*	"0.1"
"OccProfile.DIST_TRAVELLED_DOUBLE_DIST"	*<current room distance penalty>*	" 0.019"

In addition to the previous properties, the following may optionally be specified per-occupant to override the occupant shape as a cylinder:

Table 27. Optional Occupant Parameter Override

Key	Value description	Value example
"OccProfile.DIAMETER"	*<diameter of the occupant in meters>*	".4"

Key	Value description	Value example
"OccProfile.GEOM_DIAMETER"	*<geometry diameter of the occupant in meters>*	".33"
"OccProfile.HEIGHT"	*<height in meters>*	"1.8"

NOTE If both "OccProfile.DIAMETER" and "OccProfile.SHAPE" are specified on an occupant line, the diameter parameter overrides the shape. If "OccProfile.DIAMETER" is specified and "OccProfile.GEOM_DIAMETER" is omitted, the diameter also used as the geometry diameter. If "OccProfile.HEIGHT" is omitted, the default value of 1.8288 is used.

Example of an occupant line:

```
0: {"OccProfile.DIAMETER":".5","loc":"-2.017382141545266 -0.32021938754686585
0.0","OccProfile.CURR_DOOR_PREF":"0.55","profile":0,"name":"00001","OccProfile.PRIORITY_L
EVEL":"0.34669095427694063","OccProfile.DIST_TRAVELLED_DOUBLE_DIST":"0.002272613706753919
","id":0,"behavior":0,"OccProfile.REAC_TIME":"0.1","rseed":7327679609280221184,"OccProfil
e.MAXVEL":"1.51234"}
```

11.18. ELEVATORS

Elevators are defined in following sections.

The following section (optional) contains elevator information.

```
[elevators]
key value
...
```

Elevators are stored in a JSON key-value format. Following table shows available keys and values for the elevators.

Table 28. Elevators

Key	Value description	Value example
"name"	*<name of the elevator>*	"Elevator01"
"id"	*<id of the elevator>*	0

Key	Value description	Value example
"initial_floor"	*<id of the elevator room that the elevator will be at when the simulation starts>*	0
"double_deck"	*<whether the elevator is a double-deck elevator>*	false
"lower_deck_discharge"	*<lower deck discharge id of a double-deck elevator, only available for double-deck elevators>*	6
"upper_deck_discharge"	*<upper deck discharge id of a double-deck elevator, must be directly above the lower deck discharge, only available for double-deck elevators>*	7

The following section contains further elevator information.

```
[elevator-discharge]
discharge open-time close-delay size-factor max-density
...
```

Table 29. Elevator Discharge

discharge: int	index of discharge node
open-time: double	minimum time the elevator doors are open for pickup (s)
close-delay: double	delay between the last entering occupant and the doors closing for pickup (s)
size-factor: double	always 0.0
max-density: double	nominal load/elevator area

The following section contains information specific to elevator levels.

```
[elevator-level-data]
elevator-id pickup level-id open-time close-time pickup-time discharge-time delay
...
```

Table 30. Elevator Level Data

elevator-id: int	index of the elevator
pickup: int	index of pickup level node
level-id: int	index of floor
open-time: double	time it takes to open the doors (s)
close-time: double	time it takes to close the doors (s)
pickup-time: double	time it takes to go from the discharge floor to the pickup floor, excluding door open/close times (s)
discharge-time: double	time it takes to go from the pickup floor to the discharge floor, excluding door open/close times (s)
delay: double	delay for when this level can start picking up occupants (s)

The following optional section defines linked elevators.

```
[elevator-links]
<comma separated list of elevator indices>
...
```

The following optional section defines level priorities of an elevator.

```
[elevator-priority]
elevator-id, <comma separated list of elevator level indices>
...
```

11.19. OTHER NOTES

Lines beginning with the comment character ("#") will be ignored.

Lines are considered to be delimited by spaces and commas. Strings containing spaces and commas should be enclosed in double-quotes. The following three VERT definitions are identical

to inferno.

```
1, 1, 0
1.0 1. 0.
1, 1 0
```

Chapter 12. View File Format

Chapter 8 of the Pathfinder User Manual describes how to save camera positions (Views) that can be recalled later and to create Camera Tours that allow flythrough animations of a model. The data defining these views and tours is contained in the view.json file. This file uses the JSON name/value pairs and ordered list format.

Table 31. View

Name	Value description
`"name"`	Name of the view.
`"type"`	`"camera_view"`
`"state"`	Data that defines the view. This is a list.
`"loc"`	XYZ coordinates of camera
`"ref"`	XYZ coordinates of point camera is looking at.
`"up"`	Vector that defines up direction (should be orthogonal to view direction from location to reference).
`"zoom"`	Factor >0 that results from mouse scroll wheel movement.
`"zoomloc"`	Screen location of cursor when scroll wheel was zoomed.
`"near"`	Not used.
`"far"`	Not used.
`"near"`	Not used
`"fov"`	Field of view in radians.

A *Tour* object contains the following pairs:

Table 32. Tour

Name	Value description
`"name"`	Name of the tour.
`"type"`	"camera_script"
`"tstart"`	Start time of the tour
`"loop"`	Tour returns to first view.

Name	Value description
`"repeat"`	Repeat sequence.
`"crsalpha"`	Controls shape of camera path between tour views. 0.0→Uniform, 0.5→Centripetal, 1.0→Chordal.
`"node"`	Data that defines the tour. This is a list that includes the view parameters.
`"treach"`	Amount of time to reach the node.
`"twait"`	Amount of time to wait at the node.

Chapter 13. Occupant Slowing In Smoke

Pathfinder computes a maximum walking speed for occupants in smoke based on the relationships in (Fridolf et al. 2019). An occupant's maximum walking speed in Pathfinder begins to decrease linearly, starting at a 3 m visibility, until an occupant reaches a 0.2 m/s minimum walking speed. This is described by the formula:

$$w = min(w_{\text{smoke-free}}; max(0.2; w_{\text{smoke-free}} - 0.34 \times (3 - x)))$$

Where:

w = The maximum walking speed of an occupant in smoke.

$w_{\text{smoke-free}}$ = The maximum walking speed of an occupant in normal conditions.

x = The visibility (in meters) at the occupant's location as output by FDS.

Chapter 14. Fractional Effective Dose (FED)

The Pathfinder calculation of Fractional Effective Dose (FED) uses the equations described in the SFPE Handbook of Fire Protection Engineering, 5th Edition, Vol 3, Chapter 63, pages 2308-2428 (SFPE 2016). The implementation is the same as used in FDS+EVAC (Korhonen 2018), using only the concentrations of the narcotic gases CO, CO_2, and O_2 to calculate the FED value. See the FDS User's Guide (McGrattan et al. 2019) for more information about FED calculations in FDS.

$$FED_{tot} = FED_{CO} \times V_{CO_2} + FED_{O_2}$$

This calculation does not include the effect of hydrogen cyanide (HCN) and the effect of CO_2 is only due to hyperventilation. Carbon dioxide does not have toxic effects at concentrations of up to 5%, but it stimulates breathing which increases the rate at which the other fire products are taken up.

The fraction of an incapacitating dose of CO is calculated as (SFPE equation 63.18):

$$FED_{CO} = 3.317 \times 10^{-5}[CO]^{1.036}(V)(t)/D$$

Where:

CO = Carbon monoxide concentration (ppm v/v 20° C)

V = Volume of air breathed each minute (L/min). Value is 25 L/min for an activity level of light work – walking to escape.

t = Time (minutes)

D = Exposure dose (percent $COHb$). Value is 30% for an activity level of light work (walking to escape).

Hyperventilation due to carbon dioxide can increase the rate at which other toxic fire products (such as CO) are taken up. The multiplication factor is given by (SFPE equation 63.34):

$$V_{CO_2} = exp(0.1903 \times \%CO_2 + 2.0004)/7.1$$

Where:

$\%CO_2$ = volume fraction of CO_2 (v/v)

The fraction of an incapacitating dose of low O_2 hypoxia is calculated as (SFPE equations 63.27 and 63.30):

$$FED_{O_2} = \frac{t}{exp[8.13 - 0.54(20.9 - \%O_2)]}$$

where:

t = Time (minutes)

$\%O_2$ = volume fraction of O_2 (v/v)

The design of this equation has the consequence of giving non-zero values for low-oxygen hypoxia even in conditions considered to be safe. At an oxygen concentration of 20.9% (sea level), this equation gives FED_{O_2} of 1.0 after 2.36 days. At an oxygen concentration of 20.78% (FDS default AIR species), this equation gives FED_{O_2} of 1.0 after 2.21 days.

For reference, (OSHA 1995) places the low-oxygen limit for safe working conditions at 19.5% and (Purser 2010) states that "most people can tolerate hypoxia due to low inhaled oxygen concentrations down to around 11% inspired oxygen without suffering serious effects".

To prevent misleading FED accumulation on occupants that are in safe conditions, Pathfinder allows a limit to be specified in the user interface to control when the FED_{O_2} term will contribute FED – in terms of an upper bound on oxygen concentration. The default value is 19.5%.

Since the concentrations of CO, CO_2, and O_2 vary with time and space, the equations are integrated using the current exposure of the occupant to calculate FED.

The measurement location for FED quantity sampling is 90% of occupant height. For example, FED data for an occupant with height=1.8 meters would be sampled 1.62 meters above the occupant's location. When the occupant is not inside an FDS mesh, the FED calculation is paused until the occupant enters another mesh.

Chapter 15. Calculation of Measurement Region Quantities

The calculation of density and velocity in measurement regions uses an implementation of Steffen and Seyfried's Voronoi diagram-based method (Steffen and Seyfried 2010). In this method a Voronoi diagram is created to divide space among occupants. Each occupant's density is calculated based on the size of their cell in the Voronoi diagram. These densities are then combined using a weighted average, where the weights are the portion of the measurement area that intersects the Voronoi cell.

Occupants whose location is up to 1.41 meters outside the measurement region will contribute to the measurement, but more distant agents will be ignored.

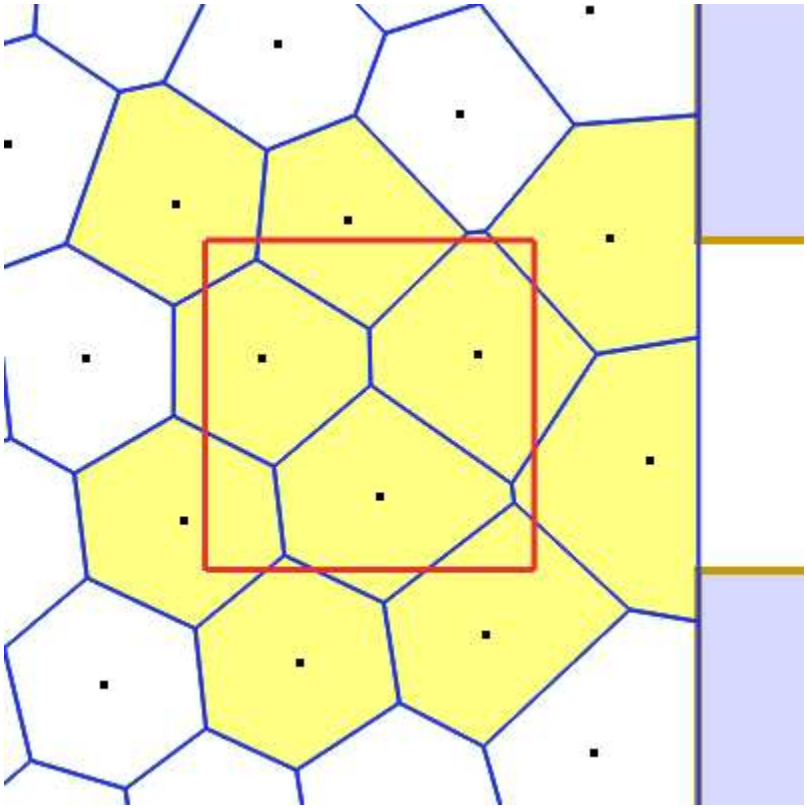

Figure 19. Intersection of a measurement region and nearby occupants' Voronoi cells.

The 1.41 meter range corresponds to a 4 m^2 square maximum area of influence for each occupant.

Bibliography

Amor, Heni Ben, Jan Murray, Oliver Obst, and others. 2006. *Fast, Neat, and under Control: Arbitrating between Steering Behaviors*. *AI Game Programming Wisdom*. Vol. 3.

Firth, Paul. 2011. “Collision Detection for Dummies.” http://www.wildbunny.co.uk/blog/2011/04/20/collision-detection-for-dummies.

Fridolf, Karl, Enrico Ronchi, Daniel Nilsson, and Håkan Frantzich. 2019. “The Representation of Evacuation Movement in Smoke- Filled Underground Transportation Systems.” *Tunnelling and Underground Space Technology* 90 (April): 28–41. https://doi.org/10.1016/j.tust.2019.04.016.

Fruin, J.J., and G.R. Strakosch. 1987. *Pedestrian Planning and Design*. Elevator World. https://books.google.com/books?id=vrckAQAAMAAJ.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. 1968. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” *IEEE Trans. Systems Science and Cybernetics* 4: 100–107.

IMO. 2016. “IMO Guidelines for Evacuation Analysis for New and Existing Passenger Ships. MSC.1/Circ.1533.” 4 Albert Embankment, London, Great Britain: International Maritime Organization.

Johnson, Geraint. 2006. *Smoothing a Navigation Mesh Path*. *AI Game Programming Wisdom*. Vol. 3.

Korhonen, Timo. 2018. *Fire Dynamics Simulator with Evacuation: FDS+Evac, Technical Reference and User’s Guide*. Sixth Edition. VTT Technical Research Centre of Finland: VTT. http://virtual.vtt.fi/virtual/proj6/fdsevac/documents/FDS+EVAC_Guide.pdf.

Kuligowski, Erica D., Richard D. Peacock, and Bryan L. Hoskins. 2010. “A Review of Building Evacuation Models, 2nd Edition.” Technical Note (NIST TN). National Institute of Standards and Technology, Gaithersburg, Maryland, USA, and Lund University, Lund, Sweden. https://www.nist.gov/publications/review-building-evacuation-models-2nd-edition.

McGrattan, Kevin, Simo Hostikka, Randall McDermott, Jason Floyd, and Marcos Vanella. 2019. *Fire Dynamics Simulator User’s Guide, NIST Special Publication 1019*. Sixth Edition. National Institute of Standards and Technology, Gaithersburg, Maryland, USA: NIST.

OSHA. 1995. “Precautions and the Order of Testing before Entering Confined and Enclosed Spaces and Other Dangerous Atmospheres.” Occupational Safety and Health Administration. United States Department of Labor: OSHA. https://www.osha.gov/laws-regs/regulations/standardnumber/1915/1915.12.

Purser, D.A. 2010. “4 - Asphyxiant Components of Fire Effluents.” In *Fire Toxicity*, edited by Anna

Stec and Richard Hull, 118–98. Woodhead Publishing. https://doi.org/https://doi.org/10.1533/9781845698072.2.118.

Reinolds, C. 1999. “Steering Behaviors For Autonomous Characters.” In *Proceedings of the Game Developers Conference 1999*, 763–82. Miller Freeman Game Group, San Francisco, California, USA.

SFPE. 2016. *SFPE Handbook of Fire Protection Engineering*. 5th ed. Springer-Verlag New York. https://www.springer.com/us/book/9781493925643.

———. 2019. *SFPE Guide to Human Behavior in Fire*. 2nd ed. Springer International Publishing. https://www.springer.com/gp/book/9783319946962.

Steffen, B., and A. Seyfried. 2010. “Methods for Measuring Pedestrian Density, Flow, Speed and Direction with Minimal Scatter.” *Physica A: Statistical Mechanics and Its Applications* 389 (9): 1902–10. https://doi.org/https://doi.org/10.1016/j.physa.2009.12.015.