Simulating Urban Pedestrian Behavior

DD143X: Degree Project in Computer Science, First Level

Spring 2013

Group 9

CSC, KTH

Wideberg, Robert (robwid@kth.se)

Wiss, Christoffer (cwiss@kth.se)

Subject: Artificial Intelligence

Supervisor: Ögren, Petter

Examiner: Björkman, Mårten

Simulating Urban Pedestrian Behavior

Abstract

This report presents a pedestrian simulator and attempts to answer questions about how well it simulates urban pedestrians and possible application areas for it. The results indicate that urban pedestrian behavior indeed can be simulated to a certain extent with the help of systems for needs and schedule generation. The simulator has potential to become a useful tool in city planning given more time spent on research on statistical models, which are focused on urban pedestrian behavior, and how they can be used in the simulator.

Simulering av fotgängares beteende i stadsmiljö

Sammanfattning

Denna rapport presenterar en fotgängarsimulator och gör en ansats till att svara på frågor om hur väl den simulerar fotgängare i stadsmiljö och möjliga användningsområden för den. Resultaten i rapporten pekar på att det i viss grad går att simulera fotgängares beteende i stadsmiljö med hjälp av system för behov och schemaläggning. Simulatorn har potential att bli ett användbart verktyg för stadsplanering givet att mer tid spenderas på forskning kring statistiska modeller gällande fotgängares beteende i stadsmiljö och hur dessa kan användas i simulatorn.

Statement of Collaboration

Both authors have worked together on the coding and report writing parts of the project[1]. An equal amount of individual work has been done to resolve small bugs in the simulator.

[1] For the project log see:
https://docs.google.com/document/d/1U1mCk87BPSacDMLXw-QoNtAiU3j9TYdHk_gpqfVqqGw/pub

Table of contents

1 Introduction

This project is centered on how to simulate pedestrian behavior in an urban environment with the use of computers. Simulating pedestrians could be useful in e.g. city planning and social studies, since it allows for observations on how pedestrians function as a collective and in conjunction with each other.

This is an interesting subject since it combines different aspects from areas such as pathfinding algorithms and graphical representation.

1.1 Purpose

The purpose of this project is to gain some insight in how well combinations of known algorithms and tools from computer science can simulate urban pedestrian behavior and how such a simulation can be applied to aid in solving city planning problems.

1.2 Problem Statement

The problem is to (graphically) simulate pedestrian behavior in an urban environment and to examine how the algorithms' different parameters affect the pedestrian-likeness in the simulation. Below follows the questions that this report aims to answer:

- How well can urban pedestrian behavior be modeled by combining well known algorithms and tools from computer science in a simulation?
- Can the simulated behavior be improved by adding individual needs and work schedules to the simulation model?
- What kind of city planning problems can be investigated using such a pedestrian behavior simulation tool?

1.3 Terms and Acronyms

Terms

Cost **–** An integer which represent some type of movement cost (could be a representation of fuel consumption, time, etc.).

Heuristic – A qualified guess, typically done by some mathematical function.

Interpolation – An estimation of data points between known data points.

Tile – Small squares that the map is divided into. Each tile is associated with a cost value.

Acronyms

BFS – Breadth First Search.

XML – Extensible Markup Language.

2 Background

This section presents some of the pedestrian simulators that are currently on the market and cover the differences between using a 2D- and 3D-model for the simulation. It will also cover some of the algorithms used for the simulation and explain why they are important.

2.1 Pedestrian Simulators

There are several pedestrian simulators available on the market today such as The Urban Analytics Framework[2] and PTV Viswalk[3]. These claim to be able to simulate crowding behavior, safety analysis, city planning and certain events such as sports events.

When constructing a simulator one has to decide whether it should be represented in 2D or 3D. The map objects and the pedestrians look more realistic in a 3D model since their original shapes are preserved. However, a 3D model will sometimes make the simulated behavior look less pedestrian-like. A 2D model leaves more room for the observer's imagination while a 3D model will trigger more nitpicking since it is more detailed and thus the observer will begin to compare it to real pedestrians to a higher degree (see *fig. 1*). This phenomenon resembles what is often referred to as uncanny valley, although in this case the observer does not feel uncomfortable but rather experience a loss of perceived realism in the simulation.

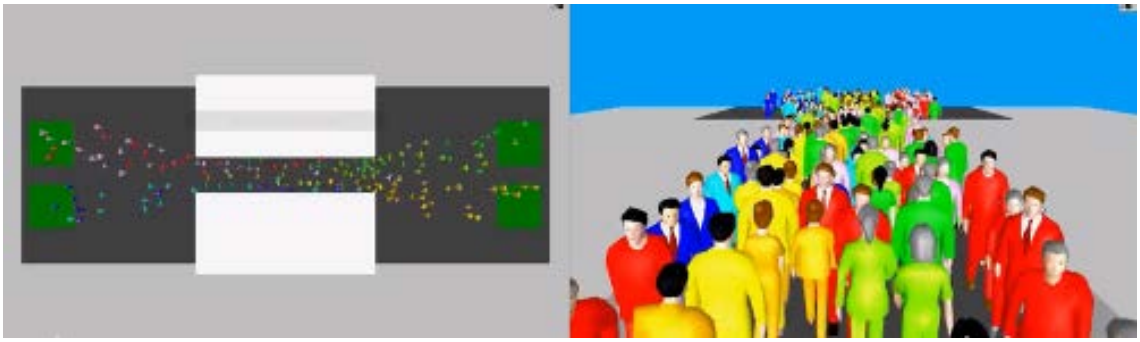

Figure 1. 2D vs 3D pedestrian simulator comparison. The crowding behavior looks realistic in the top-down 2D view but in the 3D view one can see pedestrians that are lining up against the wall and bumping into each other.

2.2 Movement Algorithms

Movement is the greedy approach to finding a way to a target. There are no guarantees that it will move along the shortest path to the target. Movement algorithms do not plan ahead but instead follow the same direction until an obstacle is found. They work well if all map tiles have the same cost values but will usually perform poorly when obstacles are introduced. However, they are not computationally expensive. An example of a movement algorithm is BFS with a heuristic. The heuristic guides the search in the direction of the target, thus minimizing the search space [1, section 1b].

[2] For more info see: http://www.pedestrian-simulation.com/

[3] For more info see: http://vision-traffic.ptvgroup.com/en-us/products/ptv-viswalk/

2.3 Pathfinding Algorithms

Pathfinding algorithms, in contrast to movement algorithms, plan ahead and are guaranteed to find the shortest path from the starting tile to the desired target. This is done by searching a large area of the map and thus pathfinding is computationally more expensive than movement algorithms. An example of a pathfinding algorithm is Bellman-Ford, which finds the shortest paths to all tiles from the starting tile [2, p. 55]. Dijkstra's algorithm is a faster algorithm that computes the shortest path to a specified target [2, p. 53]. The algorithm examines the tiles closest to the starting tile first and expands out until the target tile has been reached [1, section 1b].

2.4 The A* Algorithm

Pathfinding and movement both have strengths, pathfinding finds the shortest path and movement has a low computation cost. A combination of the two (Heuristic BFS & Dijkstra's) led to the A* algorithm [1, section 1c].

2.4.1 Overview

The A* algorithm depends on three different values:

- G – The cost from starting tile to the tile currently being examined.
- H – Heuristic, estimated cost to target tile from the tile currently being examined.
- F = G + H

Below follows a summary of the steps of the algorithm:

1. Select the tile to start from and a target tile. Mark the start tile as checked.
2. Find the adjacent tile with the lowest F cost and mark it as checked. Ignore unwalkable tiles (e.g. buildings).
    - Make the previously checked tile a parent to this new tile and store the F, G and H values for this new tile.
    - If the current tile has already been checked see if our current path has a lower cost to it (G cost). If so, change the parent tile of the current tile to the previously checked tile and update the costs (F, G and H).
3. Repeat step 2 until the target tile is reached or not found.
4. Construct the path by following each tile's parent tile from the target tile until the start tile is reached. The path is then in the reverse order that the tiles come in. [3]

2.4.2 Optimization

A* is a computationally expensive algorithm and optimizations are needed in order for the simulation to run smoothly. One important aspect of this is which data structures that are used. Instead of storing the tiles to be examined in a list, a hash table can be used to get lookups in constant time. It is also possible to check in constant time if a tile has already been visited or not by storing visited entries in a Boolean matrix. Finally, it is a good idea to store the tiles to be checked in a heap so that the tile with the lowest total cost can be retrieved in constant time [3].

The setting of the cost values is also an important factor in how the algorithm performs. It is better to set obstacles to a predefined unwalkable value instead of just setting a high value. By doing this, the algorithm avoids the obstacle tile instead of investigating whether it is a part of the shortest path.

Making the tiles larger will also boost the speed of the algorithm since it will decrease the dimension of the search space. This is however not entirely unproblematic since it reduces the precision of the paths in relation to the visual representation.

Another approach is to pre-calculate the paths on start up so that the work is already done when the simulation is running.

One must bear in mind that most of these optimizations result in a trade-off in using more memory for improved performance.

2.4.3 Heuristics

The A* algorithm depends on a heuristic to guide its search direction. There is a trade-off between speed and accuracy when deciding which heuristic to use.

One possible heuristic is the Manhattan method, which derives its name from the squared layout that is often found in American cities. The heuristic sums the horizontal and vertical distance between two tiles and thus gives an indication of how much it will cost to travel between the two tiles. This heuristic is computationally inexpensive but inaccurate in its cost estimation [3].

2.5 Bresenham's Line Algorithm

Bresenham's line algorithm was initially designed for use by digital plotters and had an advantage over similar algorithms by not requiring any multiplication or division, thus making it fast [4]. It has since become a common tool in computer graphics programming, but is not limited to only this area.

Consider a computer screen. Each point on the screen is represented by a pixel. To draw a line between two pixels on the screen a set of pixels for the line need to be decided. However, these pixels cannot, given a finite resolution, provide an exact representation of the line. Bresenham's line algorithm provides an approximation of the line by placing data points at pixels that are at the shortest distance from the line (see *fig. 2*).

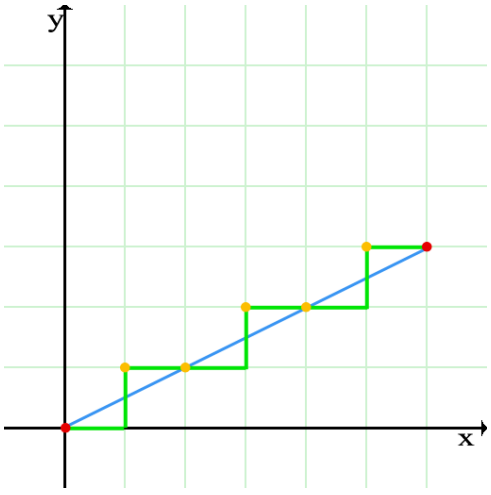

Figure 2. An approximation of a line (blue) using Bresenham's line algorithm. Yellow dots represent the resulting pixels in the approximation.

Consider an approximation of a non-steep line, i.e. a line with a larger x- than y-length, with a positive slope. The mathematical definition of a line is $y = kx + m$, where k is the slope coefficient given by $k = \Delta y / \Delta x$ and m is the y-offset. When approximating a line to discrete coordinates, the y-value is increased whenever kx turns over to a new integer. This can be done by keeping track of the numerator of the fraction and increase it by Δy for each step in the x-direction, so that when it exceeds Δx y is increased by one while the numerator is decreased by Δx. This way expensive division can be avoided while the precision of the original fraction is maintained. [5]

This is the core of Bresenham's line algorithm. If the line is steep the symmetry of the coordinate axis allows the x- and y-coordinates to be swapped in order to produce a similar non-steep situation which the algorithm can handle [6].

3 Approach

The simulator was implemented in Java and is able to display a map read from an XML format, create and handle pedestrians with varying needs and schedules and their movement along the map. The simulation is divided into a 24-hour day in order to simulate different time dependent events such as rush hours. Below follows a more in depth description of the different components of the simulator (see *fig. 3* for an overview).

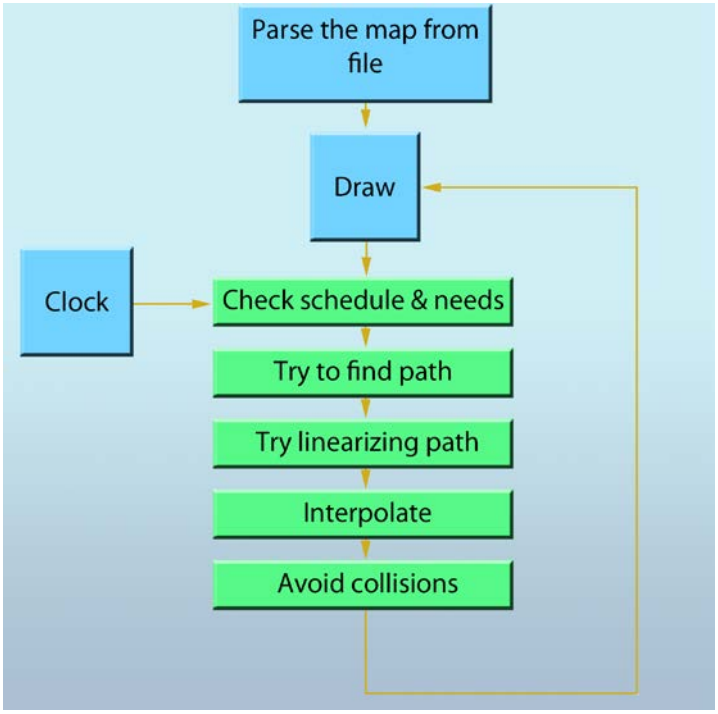

Figure 3. Overview of the pedestrian simulator.

3.1 Map

To draw the map the simulator uses a spatial database (XML) from OpenStreetMap that contains information about map objects such as buildings, streets and other urban elements. However, not all data is relevant for the simulator. A filter-tool (such as osmfilter) can be used to filter out irrelevant data from the database in order to save disk space and also slightly decrease startup time. By iterating through the data the simulator creates map objects such as buildings, ways, targets etc. The longitude and latitude coordinates of these objects are converted to screen coordinates (see *formula 1*). The graphical Java library Swing is used to draw the map on the screen.

$$\begin{cases} X = Latitude * WindowResolutionX / (MaxLat - MinLat) \\ Y = Longitude * -WindowResolutionY / (MaxLon - MinLon) \end{cases}$$

Formula 1. Conversion of map to screen coordinates. MaxLat/MinLat and MaxLon/MinLon refer to the maximum/minimum latitude and longitude that any coordinate has on the map.

3.2 Pathfinder

The A* pathfinding algorithm is used for the simulated pedestrians' pathfinding since it minimizes the search space and thus improves the performance [1, section 1b].

3.2.1 Costs

After the data has been loaded into the simulator cost values are assigned to the tiles of the different objects. These costs are stored in a matrix that represents the map divided into tiles. The pixel size of these tiles depends on the chosen map resolution.

The pathfinding algorithm will then use these costs to calculate a path through the map for each simulated pedestrian. Simulated pedestrians should avoid walking through buildings (unless there is an underlying road or target inside). This is done by making the pathfinding algorithm avoid object tiles assigned with unwalkable costs.

3.2.2 Interpolation

If each simulated pedestrian would move from one tile to another the produced motion would appear choppy with a low update rate and too fast with a high update rate. Interpolation is used to solve this. This is done by choosing an interpolation position, which could be the next tile or a position further away and then move the simulated pedestrian to this position with interpolation. First the length to the interpolation position from the simulated pedestrian's current position is calculated, this will be used to check when the current interpolation is done. Then the cosine and sine of the angle of the right-angled triangle, with points in both positions, are computed. The interpolation along the x- and y-axis is each taken from the cosine and sine scaled with the speed of the simulated pedestrian, i.e. how many pixels it should move per update. This will then produce a smooth continuous motion when applied.

3.2.3 Linearizing the Paths

In order to interpolate over large paths there is a need to linearize the path. The pathfinder produces some paths that are not realistic for urban pedestrians. The reason behind this is that the pathfinder works against a matrix with cost values that does not have the same resolution as the screen. This will cause a zigzag-behavior as the simulated pedestrian tries to walk up along a diagonally set road. To avoid this behavior a check is introduced to see if it is possible for a simulated pedestrian to interpolate from the current tile to a tile further along its path, thus producing a linear motion (see *fig. 4* for an illustration). This check uses Bresenham's line algorithm to see if it can reach another tile in the path without crossing tiles with a higher cost than the starting tile. After a specified amount of tiles have been checked, or if the check fails, the latest tile that could be safely reached is set as the interpolation position and the preceding tiles in the path are deleted. The check is then redone each time a simulated pedestrian is done interpolating between two positions.

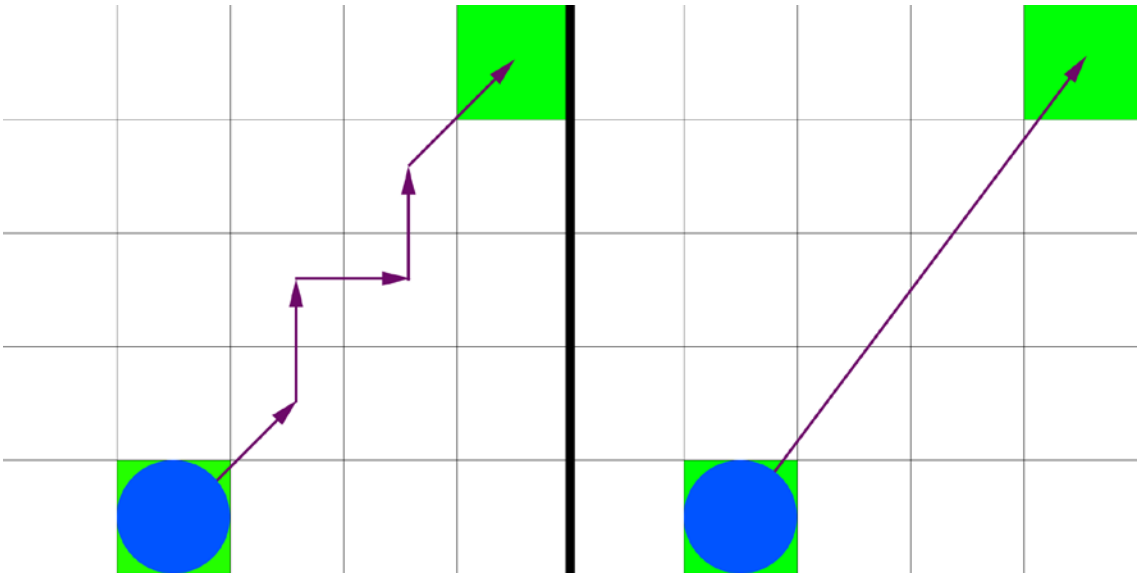

Figure 4. Comparison against movement zigzagging (figure to the left) and resulting interpolation (figure to the right).

3.2.4 Collision Avoidance System

In order to avoid simulated pedestrians walking into and through each other a collision avoidance system is used. Each time a simulated pedestrian's interpolation reaches a new tile a check is done, along its currently heading direction, to see if there is a risk for a collision with another simulated pedestrian. This works by using a prediction matrix representing the tiles and assigning an individual id to each simulated pedestrian. The tiles that each simulated pedestrian will occupy in the future are then set, in the prediction matrix, to the corresponding id of the simulated pedestrian. Each time a simulated pedestrian move into a new tile the prediction matrix is updated accordingly and a check is made to see if there is another id nearby along the currently heading path. If so an avoiding action, which is done by temporarily changing the direction to one that is appropriate to avoid the potential collision, is executed. Otherwise, if there are no potential collisions ahead the simulated pedestrian will continue along its path.

3.3 Needs and Schedule

3.3.1 Needs

All simulated pedestrians have basic needs such as eating, using the restroom, shopping, drinking coffee, disposing of garbage and withdrawing money. Different needs occur at different rates, e.g. a simulated pedestrian's need for shopping is less frequent than its need to eat. Also some needs correlate to each other such as the need to withdraw money after shopping. Finally, needs take different amount of time to carry out, e.g. eating dinner takes longer time than disposing garbage. However, all simulated pedestrians have individual variations on how often needs occur.

3.3.2 Time

In order to maintain a schedule for each simulated pedestrian a clock is necessary. Hence a 24-hour clock is implemented in the simulator and also visualized for the viewer. The latter is important since it allows the viewer to gain insight into how time progresses in the simulation. This allows the viewer to get a sense of if the simulated

pedestrians' speeds are accurate or not. When time progresses at a high rate the simulated pedestrians have a high speed and vice versa. It is also interesting for the viewer to see if a simulated pedestrian's behavior reflects the time of day. For example, at the early hours of the day there is not much activity, whilst during midday activity is higher. This allows for simulation of crowding such as the rush hours.

3.3.3 Schedule

An individual schedule is assigned to each simulated pedestrian at the start of each simulator day. A schedule consist of a start time, which is when the simulated pedestrian should appear on the map, events, which have a start time and duration, and an end time, which is when the simulated pedestrian should disappear from the map. Since an urban environment is simulated, pedestrians appear and disappear via different means of public transport, e.g. subway stations, bus stations etc. The simulated pedestrians' start and end time are generated by a normal distribution set around a fix time. Ideally a Poisson distribution would be used, since this is statistically more correct. The normal distribution is however already implemented in Java[4] and sufficient for the simulation.

In order to increase the complexity of the simulation there are different types of simulated pedestrians, each with their own set of available schedule events. Schedule events consist of activities that are not based on needs but rather on objectives. One example is students going to classes at a specified time. They typically have a longer duration than most needs and a higher priority, so that a simulated pedestrian will ignore needs when it has an occurring scheduled event.

4 Actually, the implementation, `Random.nextGaussian()`, returns an *approximation* of a normal distributed value.

4 Results and Discussion

Note that the results regarding if simulated pedestrians behave in a realistic manor are largely based on the intuitive notion of the authors. The motivation behind this is that most humans have an intuitive notion of what is considered as realistic urban pedestrian behavior.

4.1 Pathfinding

The cost based model described in section 3.2.1 enables the simulated pedestrians to prioritize certain valuable paths, while also being able to avoid certain unwalkable areas such as buildings, train tracks etc. The simulated pedestrians follow footways, which have a small cost, while at the same time stay clear of streets, which have a high cost.

The pathfinder will always find the path with the lowest cost to the target, which usually is realistic compared to how urban pedestrians want to minimize their paths while still remaining safe. However, urban pedestrians are not always familiar with an area; sometimes they walk the wrong way or at least not the most efficient way. The pathfinder in its current state cannot model this.

By implementing pedestrian crossings, which is seen as low cost areas, the simulated pedestrians are able to find a path with a lower cost across streets (see *fig. 5*). This is similar to how urban pedestrians use the crossings in order to avoid getting run over.

Figure 5. Example of pedestrians using a crossing. The snapshot to the right shows the cost matrix where black indicates unwalkable areas, red indicates areas with a high cost i.e. should be avoided if possible, green indicates areas with a low cost and white indicates neutral areas that have medium cost.

The speed of the simulated pedestrians is a difficult factor to decide. Should it be synched to the simulation's clock and how fast do urban pedestrians move? Self-observations on the time it takes to walk a certain path, present in the simulator, lead to the current setting of the simulated pedestrians' speeds. This solution is not without its flaws. The simulated pedestrians' motion can seem choppy when they cover large distances in a short amount of time. This is directly dependent on the simulation's clock; it would be possible to set the clock's default time-interval to a larger value thus producing a smoother motion but with a slower clock.

If no interpolation is used the simulated pedestrians walk choppy and zigzag along their path. This is not a behavior that occurs with urban pedestrians. Typically, what urban pedestrians do is to choose a point far off in the distance and then proceed to walk toward it. When they have reached this point they change direction toward a new point and repeat the process. This is similar to interpolation. By using interpolation in the simulator a smoother motion for the simulated pedestrians is generated.

The interpolation always interpolates to straight lines. Consider an urban pedestrian walking around a bend. This movement will take on a curvature motion rather than two straight lines. The interpolation algorithm could be upgraded to use Bézier curves to simulate this. However, its current version does generate a perceived urban pedestrian likeness since paths around buildings will result in a few short straight lines, which in turn produces a motion that is adequate.

4.2 Collision Avoidance

If the simulation is supposed to have a realistic scale the size of the simulated pedestrians must be decreased. This will however make it harder to observe individual simulated pedestrians (see *fig. 6* for a comparison). On the other hand, a smaller size gives an easier way to see how crowding occurs with the resulting effect that collision avoidance between simulated pedestrians is unnecessary. A larger size though makes it easier to observe individual simulated pedestrian behavior, but then collision avoidance is required in order to maintain realism.

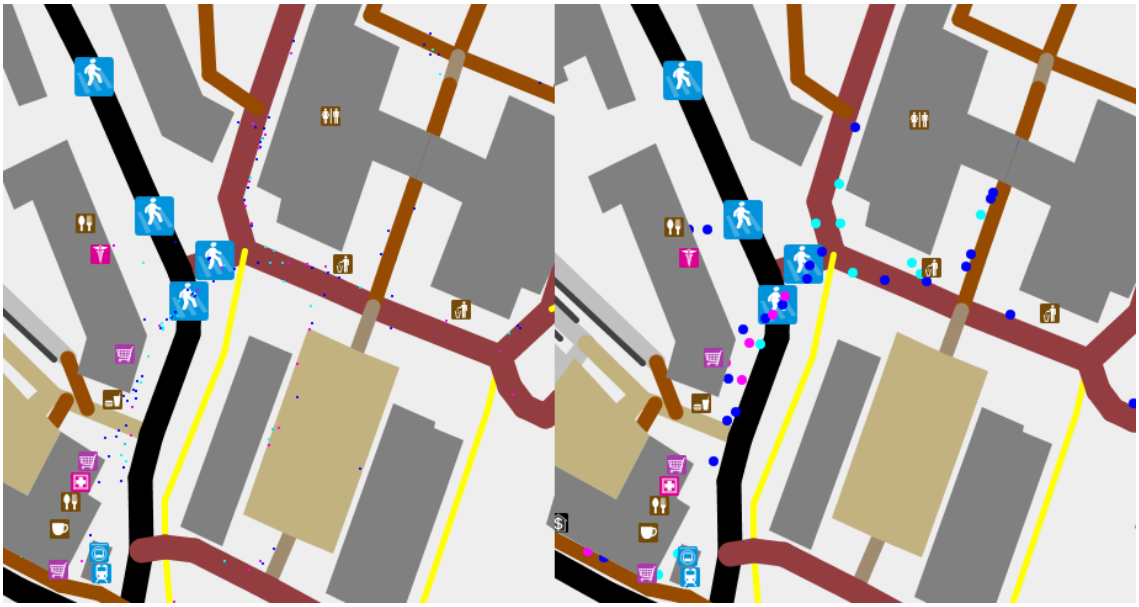

Figure 6. Comparison between using a small or large size for pedestrians.

The implementation of the collision avoidance system described in section 3.2.4 leads to simulated pedestrians that walk over streets in order to avoid collisions, a behavior sometimes seen in urban pedestrians.

To make a reasonably long avoiding maneuver is essential to increase the quality of the simulation. Urban pedestrians do short avoiding maneuvers when near a potential collision and longer if they are farther away.

Due to the collision avoidance system, situations occur that might not occur with urban pedestrians. One example is that simulated pedestrians walking alongside a street might avoid a potential collision by walking out on the street, even though they could as well

have walked in another less dangerous direction. One way to improve this is to check the cost of the avoidance direction to see if it is the smallest in situations where several avoidance directions are possible.

Another limitation of the avoidance system is that it is done on an individual level; each simulated pedestrian is only concerned about its own heading. Urban pedestrians plan the avoidance, partially instinctively, together with other urban pedestrians so that they take the same path to avoid an oncoming group.

4.3 Accuracy versus Performance

The heuristic used in the pathfinding algorithm has a great impact on its performance. The reason behind this is that the heuristic guides the search in a direction and thus decreases the search space. However, this will also affect the accuracy, i.e. whether simulated pedestrians follow the ways with the lowest costs, of the paths since the heuristic is just a qualified guess on the distance to the target (see *fig. 7*). To reach equilibrium a weight, that controls the influence of the heuristic on the pathfinding algorithm, was introduced. Observations showed that 40% of the movement cost was an appropriate value, in order to make the simulated pedestrians follow the footways, for this weight when using the Manhattan method.

Figure 7. On the left side a lower weight for the heuristic is used – resulting in pedestrians who follow the footway. On the right side a higher weight is used, improving the performance slightly, but with pedestrians that do not follow the footways.

4.4 Needs

The needs system is important since it causes simulated pedestrians to avoid repeatedly going to the same place or the same type of places. For example a simulated pedestrian that goes to the restaurant will no longer be hungry which causes it to proceed to the next need (see *fig. 8*).

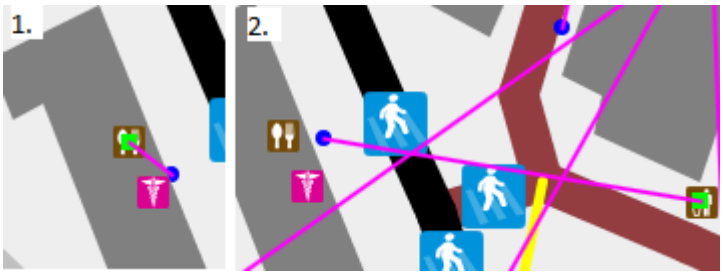

Figure 8. On the left side a pedestrian is currently heading toward a restaurant to take care of its hunger need. On the right side the pedestrian emerges from the restaurant content with its need and begins to move towards it next need, which in this case happens to be disposal of garbage.

Introducing randomness in the needs parameters results in a dynamic simulation where simulated pedestrians do not carry out needs in the exact same order. This resembles urban pedestrians who each have their own personality and thus variation in their needs. Without randomness, patterns of simulated pedestrians taking care of the same needs arise. Needs do not have a limit on the frequency of its occurrence though; a simulated pedestrian moves from need to need until a schedule event occurs. Urban pedestrians do not have this approach, they normally do not go e.g. from a restaurant to a café, to the toilet and then back to the restaurant. Inactivity or at least satisfaction needs to be modeled in order to make simulated pedestrians avoid doing the same set of things over and over again.

In order to make the simulation more realistic a simulated pedestrian that is looking for a waste bin goes to the closest bin instead of randomly selecting a bin. In contrast a simulated pedestrian that is hungry and is looking for a restaurant does not have the same constraints since urban pedestrians have different preferences when it comes to selecting a restaurant, thus randomly selecting a restaurant is an acceptable approach.

4.5 Schedule and Time

The scheduling system is needed in order to create time dependent events. By leaving a gap in each simulated pedestrian's schedule it was possible to simulate the lunch rush in the simulator (see *fig. 9*). Also, events for arriving and leaving the map are simulated and result in a map with fewer simulated pedestrians at nighttime than at daytime.

Figure 9. The lunch rush, to the left, compared to a later hour.

Urban pedestrians have a reason for their presence at a location, e.g. they might be going to the dentist or to school. In the simulator this is reflected by the events and schedule system. This leads to simulated pedestrians who will go to lectures for a certain amount of time and have breaks at which they take care of needs. Without a schedule, simulated pedestrians would only walk around and take care of needs, something that might appear to work at a first glance, but is a view that would quickly deteriorate over time.

The schedule system has randomness in assignment of schedules. This result in more dynamic simulations, e.g. students go to lectures at different hours. Since schedule events always have the highest priority, simulated pedestrians make sure that they e.g. go to lectures instead of taking care of needs. This resembles urban pedestrians, even though it might be optimistic to assume that everyone is able to keep their schedules. Behavior such as laziness, inability to keep time and taking care of important needs first is not represented in the current model. This could be introduced with a type of priority system that would differ between simulated pedestrians.

4.6 Investigating City Planning Problems

Investigations of city planning problems have limitations in the current version of the simulator. No real statistical data is used to generate a simulated pedestrian's needs and schedule but instead they are randomly generated. This can be resolved by collecting and using data about the urban pedestrians' preferences in the area being simulated, e.g. which restaurants that are most popular or which public transportation that is more frequently used.

Another limitation is that although the simulated pedestrians choose the waste bin or toilet that is closest to their position, urban pedestrians would rather choose e.g. a waste bin that is on the way to their next target. The simulated pedestrians are not affected by their environment and cannot, for example, get inspired to go shopping when they pass by a shop.

Finally, it is worth mentioning that simulated pedestrians do not block each other when they are at a target. This means that an unlimited amount of simulated pedestrians can go to for example the toilets at the same time. Given time, one could implement a queuing system so that simulated pedestrians wait for others to leave the target. Typically, all targets will then have a capacity e.g. toilets might hold only one simulated pedestrian at a time while a restaurant can hold more.

The purpose of this section is not to display statistically accurate results but rather to show the potentials of using a pedestrian simulator as a tool in city planning.

4.6.1 Waste Bins

The simulator can be used to plan where to place new waste bins in an area or to see which current waste bins that are used the most. By looking at data about the amount of visits to the waste bins, one can deduce which bins that are used the most and to what degree. From the results it is possible, for example, to see which bins that need to be emptied more often. It is also possible to add a new waste bin to the XML map data and experiment to see where on the map a new waste bin will be of most use. Below, in figure 10 and 11, the amount of visits to the waste bins made during one simulation day is displayed.

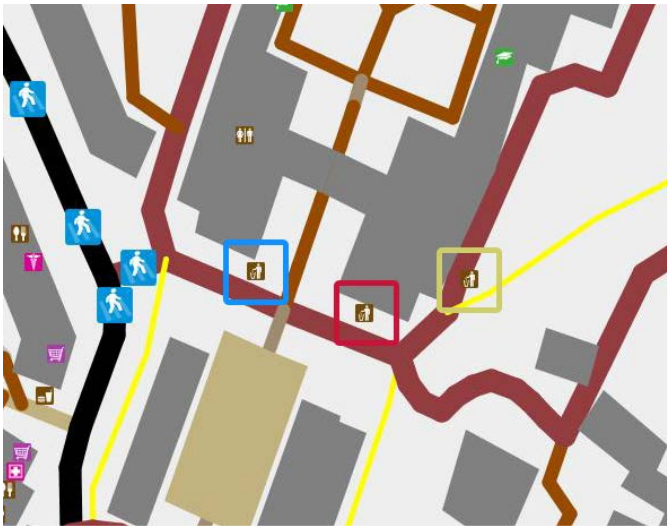

Figure 10. The three waste bins and their location on the map.

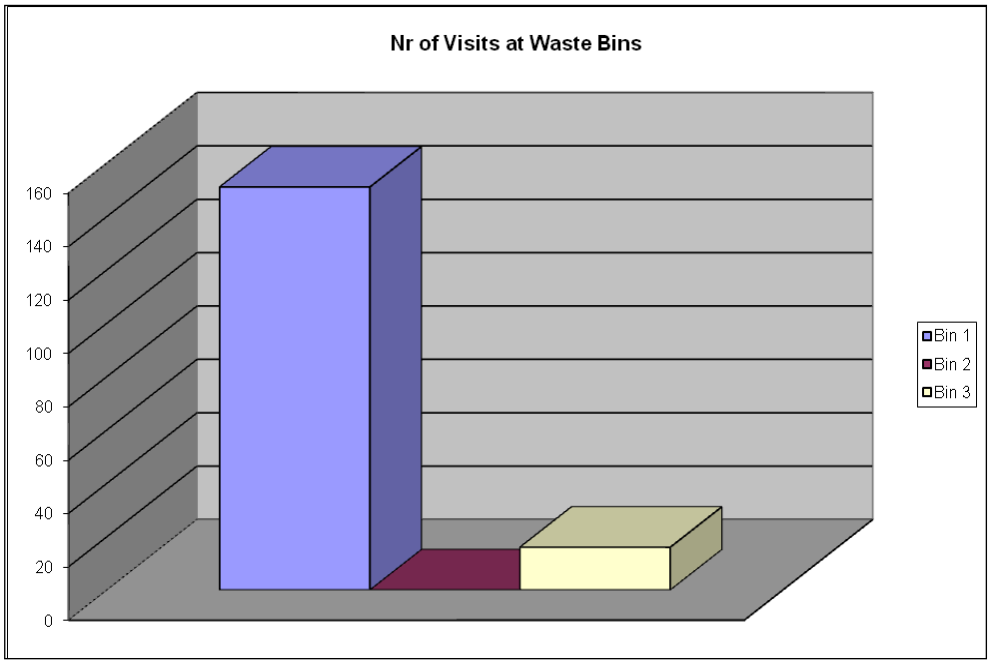

Figure 11. Total number of visits to each waste bin seen in fig. 10 after one simulation day.

It is apparent that bin #1 is being used the most and this is due to the fact that almost all the other targets are located closer to this bin than to the others. Bin #2 was never visited since it is located in between bin #1 and bin #3. In order for a simulated pedestrian to use bin #2, the need for disposal of garbage needs to rise at the moment when the simulated pedestrian is between these bins. This is not possible since the only time a simulated pedestrian's disposal of garbage need rises is at a fast food restaurant and there are none in the proximity of bin #2.

Below in figure 12 and 13 follows the results of a test of what happens when adding a new waste bin to the map. The waste bin was added near the largest collection of targets in order to see what happens to the amount of visits to the other bins.

Figure 12. New bin added to fig. 10 highlighted in cyan.

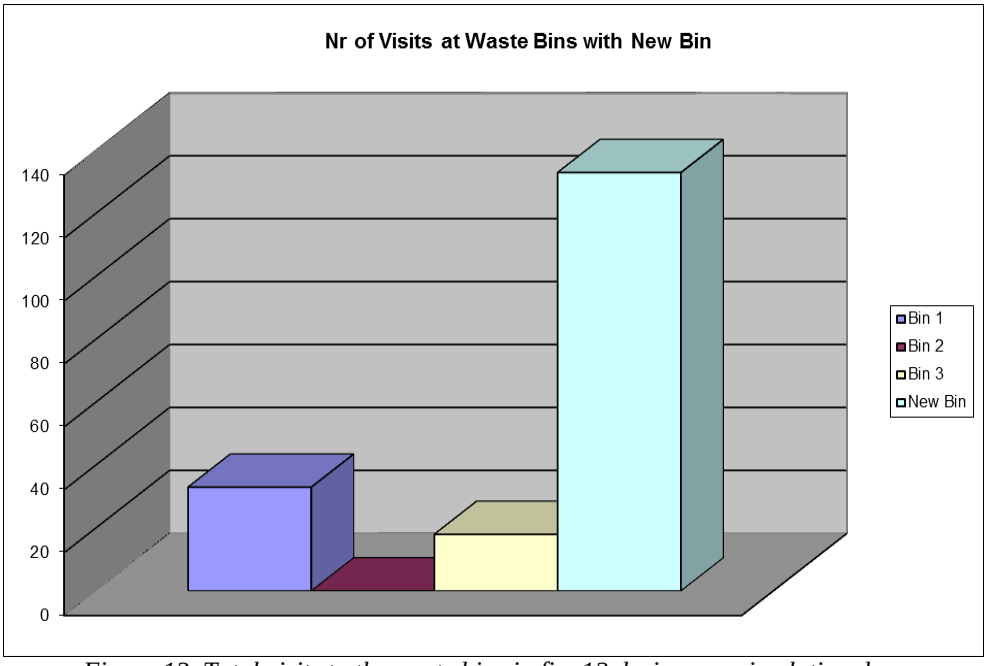

Figure 13. Total visits to the waste bins in fig. 12 during one simulation day.

The result of the test was that most of the previous visits to bin #1 are now made to the new bin. This is because the new bin is closer to the targets, which bin #1 covered, and in particular closer to the fast food restaurant. The amount of visits to bin #2 and #3 are almost unchanged since they do not cover the same area as bin #1 and the new bin.

Note that the bins have an infinite capacity in the simulation and hence they cannot be filled, therefore behavior such as going from a full to an empty waste bin is not modeled.

4.6.2 Toilets

This test investigated how the use of the toilet varied throughout the simulation day in order to see if there was any particular time when it was heavily used.

The map from OpenStreetMap that was used for the investigation had only one toilet defined. This is not realistic and surely there would be toilets in most shops and restaurants.

As noted before: toilets do not currently have any capacity in the simulator that allows an infinite amount of simulated pedestrians to use them at the same time. This is however suitable to this test since the toilet demand can easily be monitored throughout the day.

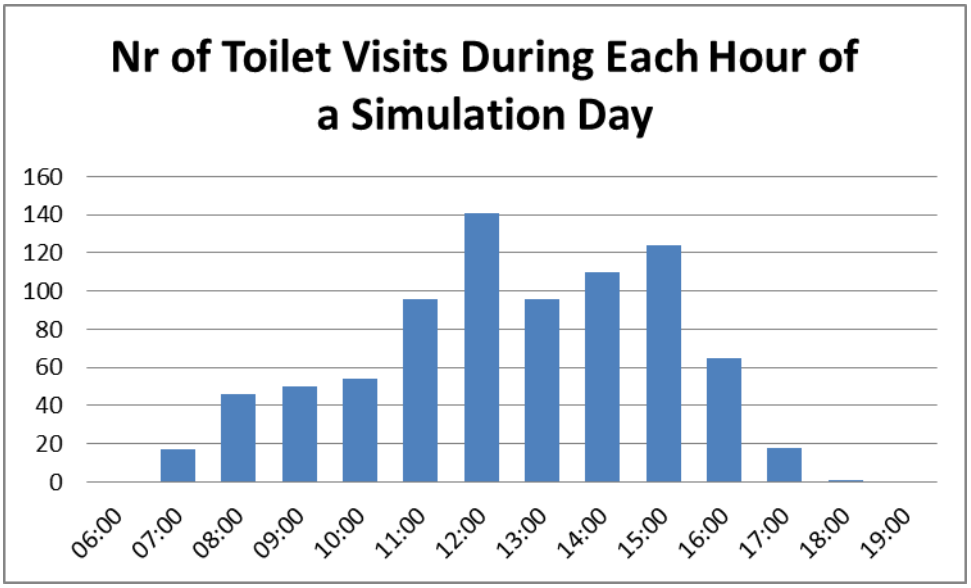

Figure 14. The number of visits to the toilet as seen during one simulation day.

As can be seen in *fig. 14* the visits are spread out over the day. However, a small peak at the lunch rush hour (around 12:00) can be seen. The reason behind this behavior is that all simulated pedestrians have a gap in their schedules around this hour and thus are free to take care of their needs. The number of visits is zero when all simulated pedestrians are at home, i.e. at morning and at night.

It is not feasible to only have one toilet for an area of this size but, as mentioned before, this is due to missing data in the map from OpenStreetMap. It is also possible to add missing map objects to the map database.

5 Conclusions

The pedestrian simulator was created in Java and is able to simulate urban pedestrians at varying speeds with a collision avoidance system. Maps used by the pedestrian simulator are read from an XML format of a specified area from OpenStreetMap. A pathfinder, based on the A*-algorithm, is used by each simulated pedestrian to find its path to a target. The simulator has the capability to simulate needs to a certain extent and also assign and maintain individual schedules.

The pedestrian simulator can to a certain extent model urban pedestrian behavior with the help of algorithms and tools from computer science. The pathfinder generates paths that are similar to paths that urban pedestrians that are familiar with their surroundings take, but has no capability to simulate urban pedestrians that are lost or unfamiliar with the area. This could be modeled with the help of a separate pathfinder, which searches a smaller space around the simulated pedestrian so its calculated paths might not be in the correct direction. Making the simulated pedestrian stop for some time could simulate uncertainty.

The collision avoidance system simulates how urban pedestrians avoid walking into each other. However, since it works on an individual level it cannot simulate urban pedestrian group behavior. Therefore the collision avoidance looks less realistic in large crowds. Making the simulated pedestrians more aware of and cooperative with each other would be a way to solve this. Another approach is to completely remove the collision handling and decrease the size of the simulated pedestrians. The viewer will then become less concerned with the individual behavior of the simulated pedestrians and focus on the group instead. The benefit of this is that it gets easier to analyze crowding behavior but it comes at the price of losing individual aspects of the simulated pedestrians.

The needs and schedule system improves the overall behavior of the simulated pedestrians. Both the needs and schedules are useful for modeling the purposes that all urban pedestrians have. Since they are generated with an element of randomness they introduce an aspect of individuality to the simulated pedestrians. The needs system keeps the simulated pedestrians busy by constantly giving them a target. However, the schedule system gives the simulated pedestrians a chance to pause from taking care of their needs by spending time at some location. The schedule system also models the fact that some activities have a higher priority than the needs of an urban pedestrian. This could be compared to how urban pedestrians prioritize their duties over needs, where a duty could be e.g. going to a class, study or research.

A weakness in the needs system is that the simulated pedestrians never are idle. Instead they are continuously fulfilling the need that is the most pressing. It would be more realistic if they only took care of needs that are at a certain critical level. If they then run out of needs they could wait or perhaps go to the next schedule event if they have any. By implementing this solution, circulating behaviors, e.g. going from a restaurant to a toilet, to a shop, to the bank and then back to a restaurant over and over again, will be avoided.

Investigations about placement of waste bins and frequency of toilet visits can be made with the pedestrian simulator. The waste bin investigation gave an insight into the importance of the placement of the bins and it also showed how it could be used to find a suitable location for a new waste bin. By investigating the usage of the toilets an indication was given of how the visits are distributed over a simulation day. This data could be used in order to see whether additional toilets are needed.

The current version of the simulator cannot simulate the behavior that occurs when the capacity of a target is filled. An urban pedestrian will search for other targets where they can fulfill their needs or wait until the target is available. If this would be implemented it is likely that the less popular targets would get more visits than they currently get.

Lastly, the pedestrian simulator shows promise in simulating urban pedestrians. Given more time and more data, about the urban pedestrians' preferences, it would be possible to implement and improve certain aspects of the simulator. This would result in a more realistic simulated pedestrian behavior and a capability to investigate more complex city planning problems.

6 References

6.1 Literature

1. Patel, Amit. Amit's A* Pages [web page online]. c1998 [updated 2013 April 8; cited 2013 April 11]. Available from: http://theory.stanford.edu/~amitp/GameProgramming/

2. Bang-Jensen J, Gutin G. Digraphs: Theory, Algorithms and Applications. London: Springer Verlag; 2000.

3. Lester, Patrick. A* Pathfinding for Beginners [web page online]. c2002 [updated 2005 July 18; cited 2013 April 11].
Available from: http://www.policyalmanac.org/games/aStarTutorial.htm

4. Bresenham J. E. Algorithm for computer control of a digital plotter. IBM SYST J 1965; 4 (1): 25-30.

5. Claridge, E. Derivation of the Bresenham's Line Algorithm [appendix online]. [updated 2013 21 March; cited 2013 April 11].
Available from: http://www.cs.bham.ac.uk/~vvk201/Teach/Graphics/Bresenham_derivation.pdf

6. Ootjers, Tom. Line Drawing Algorithm Explained [webpage online]. c2001 [cited 2013 April 11]. Available from: http://www.gamedev.net/page/resources/_/technical/game-programming/line-drawing-algorithm-explained-r1275

6.2 Figures

Fig. 1. 3D vs 2D view of pedestrian simulation.
http://www.youtube.com/watch?v=Hoq6abqb2SQ
[Accessed 11th of April]

Fig. 2. Bresenham's line algorithm.
Self-made figure.

Fig. 3. An overview of the pedestrian simulator.
Self-made figure.

Fig. 4. Zigzagging behavior and resulting interpolation.
Self-made figure.

Fig. 5. Pedestrians using a crossing.
Figure was taken from the pedestrian simulator.

Fig. 6. A comparison between using a small or large size for pedestrians.
Figure was taken from the pedestrian simulator.

Fig. 7. Image depicting the effect that the heuristic have on the pedestrians' pathfinding.
Figure was taken from an earlier version of the pedestrian simulator.

Fig. 8. Image depicting the needs model.
Figure was taken from the pedestrian simulator.

Fig. 9. Comparison of the amount of pedestrians at lunch and a later hour.
Figure was taken from the pedestrian simulator.

Fig. 10. The three waste bins present at the map used in the simulation.
Figure was taken from the pedestrian simulator.

Fig. 11. Total amount of garbage disposal at the different waste bins.
Self-made diagram based on data from pedestrian simulator.

Fig. 12. The new waste bin's location on the map.
Figure was taken from the pedestrian simulator.

Fig. 13. Total amount of garbage disposal at the different waste bins after adding a new to the map.
Self-made diagram based on data from pedestrian simulator.

Fig 14. Number of visits to a toilet, present in the map used in the simulation, during one simulation day.
Self-made graph based on data from pedestrian simulator.

6.3 Resources

Below follows links to the different resources used in the creation process of the simulator.

http://wiki.openstreetmap.org/wiki/Osmfilter
Wiki page for the osmfilter tool.
[Accessed 11th April]

http://docs.oracle.com/javase/tutorial/uiswing/
Java Swing library tutorial.
[Accessed 11th April]

http://www.openstreetmap.org/
The open source online map web project where the map data used in the simulator is extracted from.
[Accessed 11th April]

http://www.sjjb.co.uk/mapicons/
Free icons that are used for marking targets on the simulator map.
[Accessed 11th April]