Paper No. 01-2222

PEDFLOW: Development of an Autonomous Agent Model of Pedestrian Flow

Robert Kukla, Jon Kerridge, Alex Willis, Julian Hine*
Transport Research Institute, Napier University
Redwood House, 66 Spylaw Road, Edinburgh EH10 5BR, UK
Email: pedflow@dcs.napier.ac.uk

ABSTRACT

The paper discusses the need for an autonomous agent approach for the modeling of pedestrians in urban environments and places PEDFLOW in the context of existing models. PEDFLOW is a microscopic model of pedestrians' movement, where each pedestrian is represented as an agent capable of making its own decisions based upon a part of the observable scene local to that pedestrian. The model, implemented in Java, provides a framework where agents are visualised as squares in a grid and movement is modelled as a change of grid position with a delay that characterises the speed of the agent. A single rule set is utilised that is made specific to each agent by the incorporation of parameters characterising 'types' of pedestrians. The rules originate from computer aided analysis of video footage and are transformed into a form that can be efficiently processed by the agent. By adding tools to extract measures of pedestrian flow, the PEDFLOW model will be made useful to urban planners to evaluate infrastructural changes intended to promote walking in the urban environment.

INTRODUCTION

There is now a clear interest in reversing the decline in walking that has been experienced in the United Kingdom over the last twenty years. However, although some measures have been suggested to target this decline (such as improving the interchange between pedestrian and other modes of transport to provide a 'seamless journey'), the UK Department of the Environment, Transport and the Regions (DETR) continues to voice concern that little is known about which measures are most effective in encouraging walking (1). The PEDFLOW project aims to address this issue through the development of a flexible, realistic design tool that could assist urban planners to evaluate the likely efficacy of putative pavement and street designs in encouraging pedestrian activity prior to their implementation. One key aspect, for example, is the evaluation of new designs in terms of maximizing pedestrian flow in areas where congestion occurs.

The main aims of the PEDFLOW project are to: (a) explore the interactions between pedestrians and other entities in a range of urban environments, through empirical studies of pedestrian behavior 'in real life'; and (b) to develop an agent-based model of pedestrian movement that reliably mimics the movement decisions of 'real' pedestrians negotiating the walking environment. In doing so, the work will contribute to the further understanding of the effects of specific interventions and their potential to achieve desired changes in pedestrian mobility, behavior and perceptions of safety.

TRANSPORT RESEARCH AND AUTONOMOUS AGENTS

Historically network models, e.g. (2), have dominated modelling of vehicular traffic in transport research. Network approaches however are of limited value in modelling pedestrian traffic because they use a predefined number of routes (roads) from which the person/vehicle can choose, yet pedestrians in open spaces are not restricted in this way. The movement seems almost chaotic and can be determined by a myriad of factors including journey purpose and the interactions with other individuals and features of the built environment.

Models of pedestrians have been developed which seek to identify pedestrians at a strategic level (3), but they only represent routes approximately. Others represent pedestrians as continuous flow in order to estimate congestion in situations like the London Underground (4) or use a force model to simulate the attraction and repulsion between pedestrians and features of the environment (5). All these models have limited usefulness in dynamic situations where journey goals change, obstructions become visible and other pedestrians move around (such as shopping areas or bus stops). A way of overcoming these difficulties is the use of autonomous agents.

The STREETS model (6) is essentially a mesoscopic (7), agent-based model of large urban areas. Features of the environment (e.g. buildings or pavements) and pedestrians are modelled as agents and their attributes are automatically derived from several GIS data sets (e.g. socio-economic data, street networks).

Pedestrian agents emanate from gateways and move between pre-assigned way points. On their way they are ‘distracted’ by other agents and their route is modified by attributes like ‘walkability’ or ‘fixation’.

In comparison the PEDFLOW model is truly microscopic, because the aim is to model the movement of pedestrians, as they make each single step towards a pre-defined destination. Each pedestrian takes into account only that part of the urban space it can observe locally. They have no global view that is available in STREETS. Thus behaviour patterns that result from the model can be truly stated to be emergent (8).

Much of current agent-based work is related to mobile agents moving around the web extracting information from different sites. As they progress round the web the agents learn new strategies for obtaining and extracting the information required. Perhaps of more interest to this project is the work at Carnegie Mellon University (9) which explores the operation of a society of robots that operate co-operatively to achieve a common goal, in this case a game of ‘robot soccer’. Within the context of a game of soccer the robots are able to plan, develop and learn strategies that permit them to play more effective soccer. Of crucial importance in this environment is the ability to determine the actions of an adversary and develop strategies to overcome the problems posed by such an opponent. In a game of soccer, the overarching target, is to score goals within the rules of the game. This aspect is missing in the pedestrian environment that we are attempting to model, in that the pedestrians are all occupying the same space but are not adversaries in the sense of opponents in a game of soccer. Each pedestrian has its own individual goal or sequence of sub-goals which they attempt to achieve at the same time as other pedestrians moving in the same physical space. The other pedestrians are not strictly adversaries but the goals of each pedestrian are likely to conflict.

The idea behind the PEDFLOW model is to model pedestrians as agents (i.e. software objects) that can move in a virtual environment. Their behaviour is controlled by a set of rules applied to a snapshot of the environment. The rule set can be considered as a function where input variables (observation of the virtual environment) combined with parameters that are specific to a ‘type’ of person in a ‘type’ of situation are transformed into output variables (a change of location in a specific time). This cycle is repeated continuously. The sum of the individual decisions and their results make up what can be observed as movement. Multiple agents’ movements form a flow that is also observable and to which measurements can be applied. Note, although it can be interesting to observe individual agents, the primary output is the resulting flow. The properties of the flow are emergent (9) by nature because the microscopic interactions between agents only use local knowledge. No agent has access to all of the globally available data.

As with all models, PEDFLOW reduces the infinite variety of ‘real life’ aspects to a manageable amount. The limiting factor here is computational power and the number and complexity of the rules used in the decision-making process. The system in its proposed form is not self-learning, but all rules have to be developed manually and validated against naturalistic pedestrian behaviour. It is therefore important to decide which aspects are relevant and which are not, a process that is most likely dominated by trial and error. It is also an incremental approach because we keep the model as simple as possible initially and only if required are refinements be implemented.

ENVIRONMENT MODELLING IN PEDFLOW

PEDFLOW is concerned with objects in space; specifically their change of location over time. Both quantities need to be transferred into the digital domain, where they can only take on discrete values. With pedestrian modelling, many scenarios are restricted to a planar area (street, public place) so a 2D model is sufficient. Simplified shapes are used to represent any objects (called ‘entities’ to avoid confusion with OOP objects) in the modelled environment like items of street furniture and pedestrians. They correspond to the parallel projection of the entity onto the plane (figure 1a). A Cartesian co-ordinate system provides an intuitive means to record an entity’s position. Here the location of an object is specified by the x and y co-ordinate of a fixed point, usually the centre of mass. In case of mirror-symmetrical shapes this is the centre of the shape. Looking towards a computer model implementation, where efficient collision detection in a Cartesian co-ordinate system is required, all objects are modelled as rectangles (figure 1b). The location of the rectangles again can be identified through the co-ordinates of a selected/unified point within the square, e.g. the lower left corner of the shape or the centre. The smallest unit of interest in the model is a pedestrian, hence the size of the atomic entity is chosen to be the size of an average pedestrian. To accommodate more complex shapes, squares can be combined to form compound entities (figure 1c). This simplification implies that all objects are of multiple size of the atomic object. If the smallest unit in the co-ordinate system is now defined to be length of one side of the ‘atomic’ square, all entities are automatically aligned (figure 1d). As a result the modelling environment can be visualised as a grid, where grid elements can be occupied by entities that might be mobile (pedestrians) or form structures (street furniture, buildings).

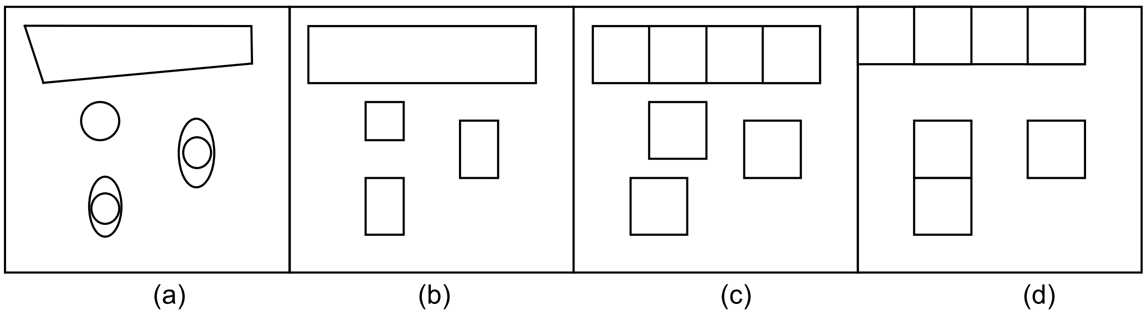


Figure 1 Levels of abstraction in a scenario involving a building, two pedestrians and a waste bin.

In order to validate the assumption that pedestrian movements can be approximated as movements between grid elements, we look at the way people move. For bipedal beings movement consists of a sequence of steps. Every step is a routine involving acceleration, side-movement and other properties that are irrelevant for modelling the overall movement. It is therefore desired to filter them out. One way is to connect the virtual footprints in sequence. This results in a series of straight lines that describes the route of the person and the time it took for each step. Turns can only occur when a foot is down, as it is the pivot point for the movement. The method has problems with time measurement and the zigzag effect resulting from the two feet. A better method is to consider the moment when both feet are next to each other and consequently the body above. Over a journey, the point between the feet at this moment gives a similar trajectory and better means of identifying the time for each 'step'.

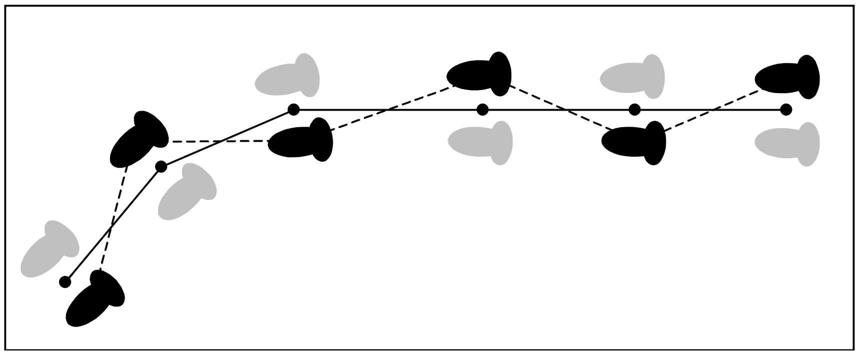


Figure 2 Using footsteps to identify the route of a person.

The result is a number of segments, where the speed is determined by the length and time it took to make the step. For modelling purposes it is desirable to unify either time or space for all entities in the model and adjust the other to compensate the resulting speed. With the former, the result is a time-clocked system, where during every time unit all modelled pedestrians move a distance that is proportional to their speed. The disadvantages here are that all entities get activated every cycle and that the granularity of the grid has to be sufficiently small to distinguish between different speeds. Both result in high computational requirements. A space-clocked system would work well in a 1D environment. Here the modelled pedestrians would move a unified distance and then 'rest' for a time that corresponds to their speed. If the origins of the entities are aligned to multiples of that distance, collision detection becomes trivial. In a 2D world however it only works for movements in x or y direction, otherwise entities would deviate from the grid positions. For diagonal movements this can be compensated by adjusting the time for the extended distance by sqrt(2). All other directions need to be approximated as a sequence of straight and/or diagonal moves. If the size of the modelled pedestrian is chosen to be a multiple of or equal to the step size, in order to simplify collision detection, the result is the same environment model as the one resulting from the earlier consideration.

The environment model can thus be reasonably represented as a 2D grid. Grid elements can be occupied by entities that are either stationary or mobile. Movement in the grid is limited to neighbouring grid elements giving a maximum of eight degrees of freedom. This is further limited to six directions with the assumption that people don't walk backwards. Movement speed is modelled as delay between position changes.

AUTONOMOUS AGENTS IN THE PEDFLOW VIRTUAL ENVIRONMENT

Autonomous agents are software objects, such as processes or class instances in an OO environment, which encapsulate local information and algorithmic expertise. They work, possibly collectively, towards a goal or a set of goals. Agents interact with each other or the environment by exchanging messages. Their actions or the results of their actions are observable. This definition makes them well suited for modelling pedestrians as they walk around. Pedestrians generally have a goal, a location or number of locations they want to visit; they observe and avoid obstructions on their way, which results in an emergent route.

An autonomous agent model of a pedestrian would need to be implemented as a software object composed of a local data structure, a means of communication and algorithms to transform observations in the virtual environment into actions. The 'goal' here is a destination location on the grid that the agent is trying to reach. In the simple case it is a grid position, but it can also be an area, possibly with prioritised grid positions. This is useful for modelling situation like queues (e.g. at a bus stop), where the goal is to get to the front of the queue or, failing that, as close as possible behind; effectively to the end of the queue. Agents maintain a list of sub goals to efficiently model the fact that pedestrians might stop on their way or, for instance, visit several shops during a shopping trip. This list can be modified through external influence, i.e. the presence of an attracting entity (e.g. shop window display). New goals can be inserted or obsolete ones removed.

Agents will move towards their goal in the most direct manner on a path approximated through steps between adjacent grid elements (including diagonal moves). A straight route however will not always be possible. Fixed objects or moving pedestrians might obstruct the path. They need to be observed and a well-defined change of direction might be required as the result. In the model, agents make that decision with every step. They do not plan a certain manoeuvre but conceive the change in an environment as a new situation with every move. An agent activation cycle can be described as the sequence of direction determination, observation, parameterisation, rule evaluation and movement.

Direction determination is the process of finding the shortest way to the goal. Mapped onto the grid and normalised to a single unit length, it represents the straight direction for the agent. It is the only absolute direction used. Observations, positions and movements are considered relative to it. PEDFLOW agents, very much like the pedestrians they represent, 'observe' their environment only in the forward direction; however not only in the grid elements straight ahead, but also in 'lanes' of grid-element width on the left and right. Currently one lane to each side is considered sufficient, but future extensions are possible. On each of those lanes an agent will look ahead until it encounters an entity in its path. The results of the observation are the category of the object and its distance for each of the lanes. Values like distance and speed are categorised according to the profile of the individual agent. The parameterised observations together with further individual agent parameters are used as input to a set of rules. Rules evaluate to a possible change of direction and a change of speed. The agent will move one step in the adjusted, determined direction and then remain inactive for a time that corresponds to its speed before the cycle starts again.

It is important to understand the distinction between 'rules' and 'parameters' in PEDFLOW. While the rules capture a general concept, the parameters modify or adjust the rules according to the individual agent's profile. Examples are measures like static awareness (SA), preferred gap size (PGS) and personal space measure (PSM) that describe the way pedestrians perceive the distance of entities. There is only one set of rules in the model that all agents use, but their individual behaviour will vary as the result of their different parameters. In the current prototype *close*, *gap*, *aware* and *far* are used as categories for distance, *same* and *faster* for speed and *same*, *opposite* and *different* for direction. For a more complete description of these parameters, please refer to (15). These categories are used in the rule, the actual range of distance/speed/direction differs with every individual agent. Rules need to be able to evaluate any possible combination of parameterised observation values for all available lanes.

A feature of the agent approach that is missing in other models, is the possibility of actively influencing other agents' behaviour. This is used to implement Attractors, where specialised agents (e.g. representing street artists or shop windows) can send messages to passing agents requesting attention. The agent will again parameterise this request and possibly adjust its path by inserting a new sub-goal. Another application is the situation where two agents face each other in a narrow space. The only way to resolve this is for the agents to swap position, an action that requires co-operation between them. Again, the active agent will send a message to its opposite requesting the swap and the other can accept or possibly take a detour.

THE IMPLEMENTATION OF THE PEDFLOW MODEL

The existence of agent toolkits like CYBELE (10) and SWARM (11) suggests their utilisation as an effective framework for the implementation of the PEDFLOW model. Investigation however revealed that CYBELE, although computational and algorithmically powerful, lacks the facilities to display agent activities in the way required for a pedestrian model. The concept of clustering agents in autonomous regions provides no advantage for the PEDFLOW application. Also one of the objectives of the project is to produce a stand-alone package for use in the urban planning process, something that is not possible with CYBELE. The modelling formalism that SWARM adopts is a collection of independent agents interacting via discrete events. No spatial information for the agents is maintained and although it could be modelled with additional agents, no advantage would be gained over an implementation from scratch. Other models show similar features that make them unsuitable for the purpose and rather to try and adapt an existing model we decided to program the model in an object-oriented language. Java was chosen of all the potential candidates, due to its portability, standardisation (Java 2.0), support for parallel processing and wealth of existing classes. The performance drawback will become less significant in the future with the availability of more efficient implementation of the virtual machine and powerful hardware. PEDFLOW uses Swing classes for its graphical presentation and user interface.

The part of the project that benefits most from the object oriented concept (apart from the user interface) is the environment model. In general terms it consists of a *simulation* class, a *grid* class, a number of *timeslot* classes and the *entity* class tree with several specialised entities. There are also a number of utility classes like *direction* and *observation* that encapsulate data structures and provide manipulation methods.

The *simulation* class acts purely as a control mechanism. It provides a way to initialise the other classes and interfaces with the GUI. By providing a well-defined interface to the GUI classes, it can be used with several different GUIs, thus making it possible to use the same model framework in an applet with little or no user control for demonstration purposes.

The *grid* class is a specialised collection class that maintains spatial information for its members. It is sub-classed from the Swing *component* class and, as the name suggests, works like a two dimensional array of entities. Entities can be added and removed, similar to the Swing mechanism for adding and removing sub-components but it is extended to include their grid co-ordinates. For performance purposes it also overwrites the *paintComponent* method to *paint* entities efficiently. The grid class provides a method for observation from a starting point in a certain direction.

Timeslot classes are used to keep track of time. They are loosely based on the bucket construct (12) but work on a higher level. They account for the fact that with a growing number of modelled entities, the maintenance of timer queues to control their periodic activation becomes computationally expensive. *Timeslots* get around this by clustering entities with the same activation time in a collection class or *slot*. These collection classes, implemented as *ArrayList*, are arranged in a circular buffer, where two neighbouring slots are activated sequentially with a delay that is equal to the smallest possible time delay in the model. The necessary size of the ring is therefore the maximum required delay divided by the minimum delay time. The *Timeslot* class maintains an internal position pointer with a method for its advancement, so the wrap-around is encapsulated. There are methods to retrieve entities from the current slot and methods to deposit entities at a positive offset to the current slot. The size of the offset gives the delay that will be experienced by the entity. Currently there are two uses for the *timeslot* class: The activation of entities already in the model and the insertion of entities at specified time during the model run. The insertion at model start is a special case of the latter.

Various specialised sub-classes are derived from the basic *entity* class, which only encapsulates the entity type and location data and, in its original form, can only be used to model immobile objects like obstructions, edges and kerbs. Its main function is to define the paint mechanism, providing a number of ways to visualise the properties of the entity through use of colour and shape. *Mobile* is a subclass of *entity* that implements movement. Speed and direction of the movement are predefined; it can be used to model moving entities where the behaviour is not of interest but their influence on pedestrians like for example passing cars. Unlike the *entity* objects, instances of that class are activated in regular intervals. *Mobile* is also the super class for the class *person* that is used to model pedestrians. The implemented rule evaluation algorithm for the agents is discussed in the next section.

The possible high number of involved agents and the requirement for at least real-time performance places high demands on the hardware used. Efficiency in the implementation of the algorithm and the possible use of multi-processor hardware help to make the application more scalable. Recent implementations of the Java virtual machine provide 'native threads' as a convenient means to efficiently utilise multiple processors. Although the autonomous agent concept in general suggest their implementation as parallel processes, in the case of PEDFLOW where the activation is space-clocked, the number of agents that are active at the same time is rather small - tests show an average of one to two agents per *timeslot* with the current implementation. Taking also into the account the additional overhead for scheduling and data locking the performance gain becomes questionable. In terms of functionality there is no justification for a parallel implementation either, as the choice which agents are activated at the same time is arbitrary and has no relation to the modelled pedestrians. Moreover it is not guaranteed that agents who require communication between them (e.g. to negotiate passage) will be in the same *timeslot*. Instead interaction between agents is implemented through forced activation of agents by the agent that initiates the interaction. The only potential for parallel processing exists for the visualisation. By moving the display-related code onto a different processor, the first CPU can be better utilised by the model. The performance impact will be evaluated in the near future, when a multiprocessor machine will be available.

RULES AND THEIR IMPLEMENTATION

The mechanism that transforms the parameterised observation into an action can be implemented in various levels of abstraction. They range from AI scripts that are interpreted through an Expert Systems class, similar to Jess (13), over manually constructed decision trees to as low a level as a lookup tables that hold all possible input combinations together with their result values. The choices differ significantly in their computational efficiency. In PEDFLOW the decision making process is performed in every agent-activation. Therefore an efficient implementation would greatly benefit the overall performance. On the other hand it is essential that the rules are written in a way that makes them easy to understand and modify and that a way exists to ensure that the rule set is complete.

Implementing behavioural rules as interpreted scripts is too inefficient. An expert system implementation that pre-compiles rules to code which can be efficiently parsed or even directly executed however would be the best balance between user-friendliness and speed. Development of such a compiler however is an extensive task and currently outside the scope of the project. Decision trees, as an alternative low-level approach, are difficult to manually construct and maintain and a complete lookup table is too big in volume. A compromise would be a sequential rule table, where the number of rules is reduced through either the use of wildcards for input variables that do not contribute to the result ("don't care" inputs) or where similar input configuration are merged. Here input variables that have the same effect on the result of the rule are replaced with an alias that covers them all, resulting in a smaller number of rules. As the rule table is subject to change in the development phase, the optimisations need to be reversible and ideally be automated.

A first step to reduce the number of required rules is to reduce the number of input variables. As described in the previous section the input variables are the entity type and, in the case of a person, its direction and speed combined with its distance. These inputs exist for the three lanes *left*, *straight* and *right*, which leads to roughly (4 types * 3 directions * 3 speeds * 3 distances) ^ 3 lanes = 1.126 million possible combinations which need to be mapped to (4 direction choices * 2 speeds + 3 avoids + 1 pause) = 12 possible outputs.

The interesting fact here is that direction and speed is only applicable for mobile entities. That makes it possible to merge that information into the *entity* type by replacing the entity *person* with *person moving in the same direction*, *person moving in a different direction* and *person moving in the opposite direction*. If behavioural evidence indicates that further discrimination is required, *person passing left-to-right* and *person passing right-to-left* can be added. Speed is only relevant for people moving in the same direction. Instead of having all possible combinations, it can be used as a criteria to split the entity *person moving in the same direction* into *person moving slower* and *person moving faster*, where the latter includes people moving at the same speed. As a result there are now 7 entity types in the model: person (same direction, faster) F, person (same direction, slower) S, person (opposite direction) O, person (different direction) D, edge E, blockage B, kerb K that can appear in 3 possible distance ranges (close C, gap G, aware A) on 3 lanes. There is also the situation to consider, where there is no entity (vacant V) on the lane within the observation range giving an overall number of possible situations of (7 entities * 3 distances + 1 vacant) ^ 3 = 10648. Rule outputs can be categorised in a similar manner as a combination of 4 directions (left L, straight S, right R, choice C) and 3 speed-choices (full F, match M, avoid A).

Although greatly reduced, it is still neither feasible nor desired to generate this number of rules manually. The solution is to use sets of entity/distance combinations in the condition part of the rule and thus merge rules that have the same outcome into one. For example, if the left and right lane are vacant (V), but there is either a blockade or a slow moving person in front of the agent the result would be move either left or right (choice C) at full speed (F). In a notation where the entities and their distance are denoted as two letters, the input values from the three different lanes connected by a Plus in the order left, straight, right and the resulting distance/speed combination is pointed to by an arrow, the rules can be expressed as:

V + BG + V → CF
V + SG + V → CF

These two rules can be merged using sets as inputs, where the left-lane and right-lane set only contains the vacant entity (V), but the straight-lane set has the elements slower person (S) and blockage (B) at gap (G) distance as shown in:

[V] + [BG, SG] + [V] → CF

Thus set-based rules cover all possible combinations of elements from the three lanes. They have to be constructed very carefully to avoid creating unwanted combinations of elements as side-effects as shown in the next example. The rule

FF + BG +FF → CF

has the same result and is therefore a suitable candidate for merging with the set based rule from before. However the resulting rule

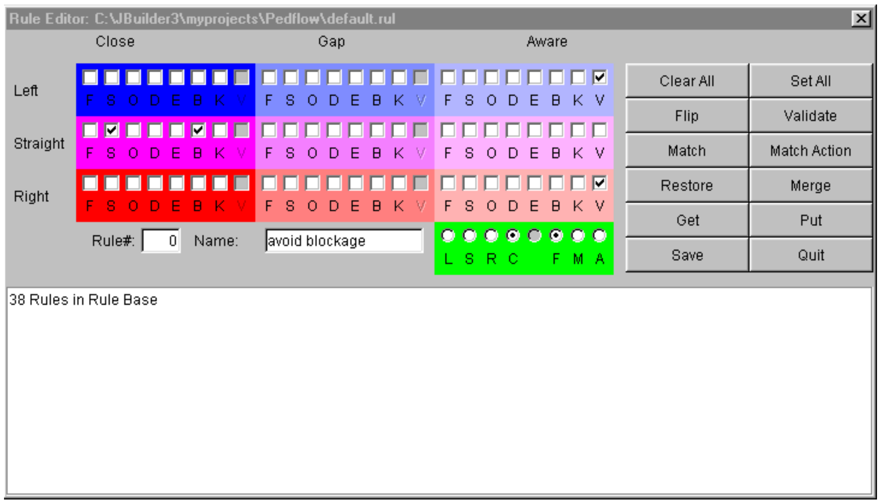
[V,FF] + [BG, SG] + [V,FF] → CF

also implies the rules

FF + SG + FF → CF
FF + SG + V → CF
V + SG + FF → CF
FF + BG + V → CF
V + BG + FF → CF

as a result of possible combinations of entities in the three lanes. In general the number of single entity rules which a set-based rule covers is equal the product of the sum of the elements in each of the lanes and can become quite large. It is important to make sure that the additional rules are correct. Under certain circumstances, this might not be the case and rules cannot be merged. Or alternatively, the resulting rule may require splitting in order to remove included simple rules that are found to be incorrect. Splitting (as the reverse of merging) has the undesired effect of removing other rules, which have to be covered separately. Generally, different grouping of elements leads to different numbers of rules and research is currently being undertaken by us to find algorithms to optimise the rule set automatically. The minimum number of rules is given by the number of possible outcomes (4 directions * 3 speeds = 12). Experience shows that for a realistic, usable rule-set, which covers the most commonly occurring situations, 30-50 rules are required. A complete rule-set, which contains all possible input combinations, is estimated to consist of over 80 combined rules.

Adding and optimising rules is an extensive, manual task. To help in this undertaking a rule-editor has been developed (figure 3). It visualises the three input sets as areas of equal colour, where the existence of a set-element denoted by the initial letter can be indicated with a tick-box. The result is displayed in a similar way in a different area as the choice of direction and speed. The software offers a number of utility functions via buttons. In addition to basic functions (like store, select and retrieve rules from the rule-set), it also provides functionality to automatically flip rules to generate the symmetrical rule, merge rules and most importantly check the validity of an edited rule against the rule-set before it is incorporated. The rules that are generated are very compact; each of the three input sets is stored as an integer with bit positions corresponding to the existence of entities in the set. Evaluation is very efficient as it is implemented as binary operations on these integers. A match is made when for all lanes the *BINARY AND* of *entity* and *input-set* evaluates to a number not equal to *zero*. If no match can be found for a given situation during a model run, the software will open the rule editor with the rule input represented in the tick boxes and prompt the user to enter an appropriate output. The resulting rule can then either be stored or merging with other rules can be attempted. Once the rule-editor is closed, the model will continue to run with the new rule in place or alternatively it can be stopped and restarted.


Rule Editor: C:\JBuilder3\myprojects\Pedflow\default.rul

	Close	Gap	Aware
Left	F S O D E B K V	F S O D E B K V	F S O D E B K [✓ V]
Straight	F [✓ S] O D E [✓ B] K V	F S O D E B K V	F S O D E B K V
Right	F S O D E B K V	F S O D E B K V	F S O D E B K [✓ V]

Rule#: 0 Name: avoid blockage

L S R [✓ C] [✓ F] M A

38 Rules in Rule Base

Figure 3 Rule Editor.

VALIDATION OF RULES AND PARAMETERS

The PEDFLOW will only be of value as a planning tool, if the rules and parameters used in the model encapsulate 'realistic' pedestrian behavior. Therefore the project is also concerned with the empirical assessment of how pedestrians negotiate their environment 'in real life'. The broad aim of the research is to characterize the different types of interaction that occur between pedestrians and other 'entities' in the natural environment (14). Currently, attention is focussed on: (a) characterizing the most common behaviors seen in various environments (such as 'overtaking' or 'yielding' to others); (b) quantifying the range of values for the model parameters (such as DWS, or PSM) associated with these behaviors; and (c) determining whether the current parameter and rule sets are sufficient to encapsulate the majority of interactions within a range of pedestrian environments.

The microscopic movement patterns of individual pedestrians are assessed primarily using observational techniques, although questionnaires and in-depth interviews will also be used to gain data on non-observable attributes (such as trip purpose, movement intentions, or perceptions of the environment). Pedestrians are filmed discretely as they negotiate short sections (12-20 metres) of city centre pavement using video cameras positioned above the area of interest. The resulting video footage is converted into digital format and imported into a proprietary image-tracking program. The image is carefully calibrated according to the 'real world' co-ordinates of the environment under analysis, taking into account any deformations arising as a result of the camera not being directly overhead the pavement. Pedestrians are selected for image-tracking, and their absolute x-y positions in space on each frame tracked and exported into 'GridTool' – a custom-developed software package designed to yield accurate measurements of distances between any two objects, and walking speeds of individual pedestrians over any distance. For example, this software can be used to estimate the distance from an obstruction at which a pedestrian starts to deviate from his or her 'straight-on' direction, or the distance between passing pedestrians at the time they are shoulder-to-shoulder. From these, values for the model parameters can be deduced (in the above example PGS, and PSM, respectively), and the extent to which factors such as the pedestrian's age, or groups size influence these values explored.

CONCLUSIONS

PEDFLOW is an autonomous agent based model of pedestrian flow. It differs from other models in that it is truly microscopic and emergent. Autonomous agents (representing pedestrians) re-evaluate their environment (static entities and other agents) with every step as they negotiating an obstructed area (e.g. pavement) and their routes emerge as a result. PEDFLOW uses a single set of rules that is made specific to individual agents by means of parameters. The rules and parameters used in the evaluation are derived from real pedestrians using a combination of qualitative and quantitative observational techniques and therefore considered a reliable basis for valid agent behaviour. The implementation in Java is compact, flexible and efficient as it avoids unnecessary functionality present in available agent-based model frameworks. It is consequently object-oriented and uses a class hierarchy to implement specialised agents (e.g. Attractors). The PEDFLOW application is executable on any platform with a Java run-time environment.

Although a prototype implementation of the PEDFLOW model exists, it needs to be extended with utilities to aid the set-up of experiments and the extraction of results, before it can be used as a tool in urban planning. Such utilities include a graphical environment editor, a means to generate an agent population based on statistical data and tools to extract Fruin measures of service. There is also scope for further research with regards to the representation of rules and their automatic optimisation. The analytic research to determine the rules and parameters that need to be applied in the PEDFLOW model is still at an early stage, but an automatic mechanism for extracting positional information has been developed.

ACKNOWLEDGEMENT

This work is funded by the UK-EPSRC grant reference GQ/M59792, awarded to Jon Kerridge and Julian Hine. It is being undertaken in collaboration with the City Council of York and Edinburgh and the consultancy Halcrow Fox, UK. We would like to thank Professor Jim Wentworth for his valuable comments.

REFERENCES

1. Department of the Environment Transport and Regions (DETR). Encouraging Walking: Advise to Local Authorities. London, March 2000, 99AILT0220
2. Wang, Y., and D. Prevedouros. Synopsis of Traffic Simulation Models. Presented at the Transportation Research Board 1998 Annual Meeting, 1997
3. Timms, P. Putting pedestrians into network planning models. Presented at the 6th World Conference on Transport Research in Lyon, 1992
4. Annesley, T., M. Dix, A. Beswick and P. Buchanan. Development and application of pedestrian assignment models in London railway station studies. *Traffic Engineering Control*, July/August 1989
5. Helbing, D. A mathematical model for the behaviour of pedestrians. *Behavioural Science*, Volume 36, 1991
6. Schelhorn, T., D. O'Sullivan, M. Haklay and M. Thurstain-Goodwin. STREETS: an agent-based pedestrian model. Presented at Computers in Urban Planning and Urban Management, Venice, 8-10 September, 1999
7. Hoogendoorn, S.P. and P.H.L. Bovy. Gas-Kinetic Modelling and Simulation of Pedestrian Flows. Presented at the Transportation Research Board 2000 Annual Meeting, 1999
8. Fisher, D.A. and H.F. Lipson. Emergent Algorithms – A New Method for Enhancing Survivability in Unbounded Systems. *The Proceedings of 32nd Hawaii International Conference on System Sciences*, (CD-ROM)IEEE, 10 pages, ISBN 0769500013, 1999
9. Stone, P. *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*, MIT Press, ISBN: 0262194384, 2000
10. Erol, K. A Study of Agent-based Traffic simulation. Final Report, FHWA, US DOT, 1998
11. Minar, N., R. Burkhart, C. Langton and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations. 1996. http://www.santafe.edu/projects/swarm/overview/overview.html. Accessed Jul. 15, 2000.
12. Kerridge, J., P. Welch and D.Wood. Synchronisation Primitives for Highly Parallel Discrete Event Simulations. Presented at HICSS-99, 1999
13. Friedman-Hill, Z E. Jess, the Java Expert System Shell. 2000. http://herzberg.ca.sandia.gov/jess/. Accessed Jul 15, 2000.
14. Willis, A., R. Kukla, J. Hine and J. Kerridge. Developing the Behavioural Rules for an Agent-based Model of Pedestrian Movement. Presented at the 25th European Transport Congress, Cambridge, Sept 2000.