D5.3.2 Final simulated users

Morgan Fredriksson, Jürgen Königsmann,

Johan Boye

Distribution: Public

Spacebook

Spatial & Personal Adaptive Communication Environment:

Behaviours & Objects & Operations & Knowledge

Deliverable 5.3.2

2013-08-31

Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development

The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	270019
Project acronym	Spacebook
Project full title	Spatial & Personal Adaptive Communication Environment: Behaviours & Objects & Operations & Knowledge
Instrument	STREP
Thematic	Priority Cognitive Systems, Interaction, and Robotics
Start date / duration	01 March 2011 / 36 Months
Security	Public
Contractual date of delivery	M30 = 2013-08-31
Actual date of delivery	2013-08-31
Deliverable number	D5.3.2
Deliverable title	Final simulated users
Type	Report
Status & version	1.0
Number of pages	13
Contributing WP	5
WP/Task responsible	LM
Other contributors	KTH
Author(s)	Morgan Fredriksson, Jürgen Königsmann, Johan Boye
EC Project Officer	Franco Mastroddi
Keywords	Simulated user, dialogue system

The partners in Spacebook are:

Umea University	UMU
University of Edinburgh HCRC	UE
Heriot-Watt University	HWU
Kungliga Tekniska Högskolan	KTH
Liquid Media AB	LM
University of Cambridge	UCAM
Universitat Pompeu Fabra	UP

For copies of reports, updates on project activities and other Spacebook-related information, contact:

The Spacebook Project Co-ordinator:
Dr. Michael Minock
Department of Computer Science
Umea University °
Sweden 90187
mjm@cs.umu.se
Phone +46 70 597 2585 - Fax +46 90 786 6126

Copies of reports and other material can also be accessed via the project's administration homepage, http://www.Spacebook-project.eu

Contents

Executive Summary 1
1. Introduction 2
2. Simulated pedestrians 2
3. Cognitive module 3
4. Spatial module 4
5. Communication between Cognitive and Spatial modules 4
6. GUI 5
7. Visibility calculations 8
8. Walk-path calculations 10
9. Final remarks 12
10. References 13

Executive Summary

This report describes the implemented simulated users for the Spacebook domain. These simulated users are capable of simulating movement in any city covered by OpenStreetMap, to understand natural language route instructions, to ask for information, and to pursue both long-term and short-term goals.

1. Introduction

This report describes the implemented simulated users for the Spacebook domain. These simulated users are capable of simulating movement in any city covered by OpenStreetMap, to understand natural language route instructions, to ask for information, and to pursue both long-term and short-term goals. In order to do this, the simulated users are implemented as a set of communicating modules: a spatial module performing collision detection, visibility calculations and low-level planning of walk-paths; a high-level cognitive decision-making module carrying out the natural-language dialogue as well as generating and maintaining the simulated pedestrian's goals; a natural-language parser and generator; and a graphical interface.

To be useful, a simulated user should accurately imitate a real user in some important regard. In order to do this, it is necessary to define a model of the user by a set of state variables that determine the state of the simulation, a set of actions the simulated user can perform, a set of inputs it can interpret, and a transition function that map state-input pairs to state-action pairs. This transition function may itself be learned from data or it may be hand-coded; in either case, the produced behaviour of the simulation should be realistic in some sense.

In a bigram model, as used by Eckert et al. (1997), the state of the simulation is determined uniquely by the dialogue act[1] of the system's latest utterance. Such bigram statistics can be learned completely from data. However, as pointed out by Schatzmann et al. (2005) this approach often leads to nonsensical dialogues. A number of researchers have therefore sought to improve on this basic model by enriching the state representation (see e.g. Levin et al 2000, Götze et al. 2010). Another problem with the bigram approach is data sparsity; not all realistic dialogue behaviours will be present in the original data set. T

Another use for simulated users in dialogue systems is as a tool for testing during dialog system development was suggested by Ai and Weng (2008). Replacing real users in early phase testing where the system to be tested is not yet stable enough for evaluation by real users.

2. Simulated pedestrians

The simulated user (henceforth referred to as "S") acts as a user taking instructions from a system giving natural-language route instructions. S can either be connected to such a system, or be controlled by a human operator entering natural-language instructions from the keyboard.

[1] A dialogue act is an abstraction of an utterance, representing its pragmatic-semantic function (see e.g. Traum (1999)).

3. Cognitive module

In its cognitive module, S maintains a list of long-term goals; each such goal is the identifiers of a place to visit. These places can possibly be very far away from S's current position. As S moves through the city, S will continuously check if its current position is close to one of those places; this mechanism mimics the desire to visit certain specific sites in the city. Furthermore S keeps a list of nodes or node types that will, once such a node is in S's view, influence S to approach it. We call such nodes attractors; these are used to simulate opportunistic behaviour (such as approaching restaurants when they come into view, perhaps to check out the menu).

Apart from long-term goals and attractors, S also maintains a stack of short-term goals. Again, each such short-term goal is an identifier of a node to go to, but short-term goals are close to S and are within S's view. Short term goals are usually put on the stack as a result of instructions from the route-giving system or human operator. However, if S receives no instructions, it will try to guess an appropriate next short term goal and put it on the stack on its own initiative. Most often, S will continue walking in roughly the same direction as before, but with a small probability S will deviate from its current course and randomly select a new direction.

In addition, S has an internal scalar representation of how assertive it is that the current direction is correct. This assertiveness is influenced in the positive direction if a given instruction can be interpreted sensibly, e.g. when the instruction is "Turn left", and it is indeed possible to turn left at S's next short-term goal, or if the instruction is "Go towards Starbucks", and the reference "Starbucks" can be mapped to a node in S's view. If it is not possible to turn left, or if there is no Starbucks in view, assertiveness will be decreased. S's assertiveness is also slowly decreased as time elapses without it having received a route instruction, and even more so if S needs to change direction on its own initiative.

S uses a natural-language parser to translate instructions into a formal meaning representation language (MRL). The reference resolution step, which interprets context-dependent references like "left", "the museum" or "Starbucks", operates on the MRL by replacing lambda-bound variables for identifiers of actual nodes. This reference resolution process is described further in Deliverable 2.3.2, section 4 (Albore et al. 2013).

S employs a probabilistic finite-state automaton to control its dialogue behaviour. Dialogue acts that can be expressed are requests for directions ("Directions to Camera Obscura"), requests for instructions ("Where should I go now?"), answers to specific questions (e.g. "Can you see Starbucks?"), acknowledgements ("Okay"), reports of miscommunication (e.g. "I didn't understand that", "I don't know where Starbucks is."), reports of success ("Thanks, I can see Camera Obscura"), and a few others. In particular, S will request an instruction when its assertiveness falls below a given threshold.

Finally, S maintains a representation of the set of currently visible objects. This is repeatedly retrieved from the spatial module (see next section).

4. Spatial module

In order for the S to produce believable behaviour it needs to be able to perceive and react to the surrounding environment. The Spatial Module creates a model of this environment by parsing an OpenStreetMap XML file. Such XML files can be generated on the fly by indicating a region on the map in the OpenStreetMap web interface (http://openstreetmap.org), and consist of lists of nodes defining a single lat-lon point, ways encoding roads, areas, buildings, etc., and relations between ways. Because of this set-up, the Spatial Module is portable between different cities without any re-configuration whatsoever.

From the OpenStreetMap XML file, the Spatial Module builds an object-oriented representation which forms the basis for all ensuing calculations. In particular, buildings and areas are represented by polygonal objects. If the height of a specific building is known, this height will be imported into the object model; otherwise the building is set to a general height value that can be specified per city or area.

5. Communication between Cognitive and Spatial modules

To produce realistic behaviour, a simulated user should not be omniscient concerning the location of all buildings, streets and landmarks in the ‚city. Therefore, the full object-oriented model of the city maintained by the Spatial Module is not directly available to the Cognitive Module. Rather the Cognitive Module can query the Spatial Module for visibility information: The query consists of two (lat,long)-positions A and B, and the response consists of a “yes”- or “no”-answer. In the case of a “yes”-answer, some extra information, some extra information is provided as well (see section 3.5).

As mentioned above, the Cognitive Module decides the next short-term goal G, i.e. the next node to go to. However, the Cognitive Module does not concern itself with the exact route to G; rather it sends a “Next-node” message to the Spatial Module, which then computes a detailed walk-path to G (see section 3.6). The Spatial Module then generates the actual movement as a stream of simulated GPS coordinates which is sent to the route-giving system that instructs the simulated user. Also the Cognitive Module listens to this coordinate stream in order to know when G has been reached, and its time to decide on the next short-term goal.

The Cognitive Module can also send other commands to the Spatial Module to control the behaviour of the simulated user (“stop”, “start”, and “reset”, “increase/ decrease speed”).

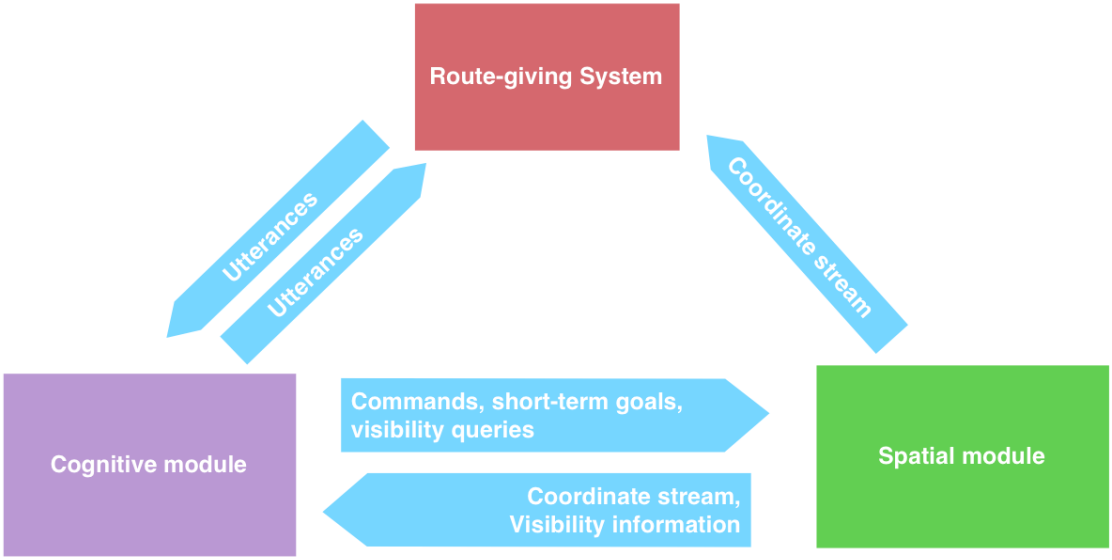

Figure 1: Communication between modules of the simulated users and surrounding systems.

6. GUI

The simulated user can be monitored from a graphical user interface. The current position is indicated by a pulsating double ring. Previous positions are shown tracing a path of the short term navigational goals passed by the simulated user. The speed of S can be changed, either manually using the "+" and "−" keys on the keyboard or by a command from the cognitive module to the spatial module.

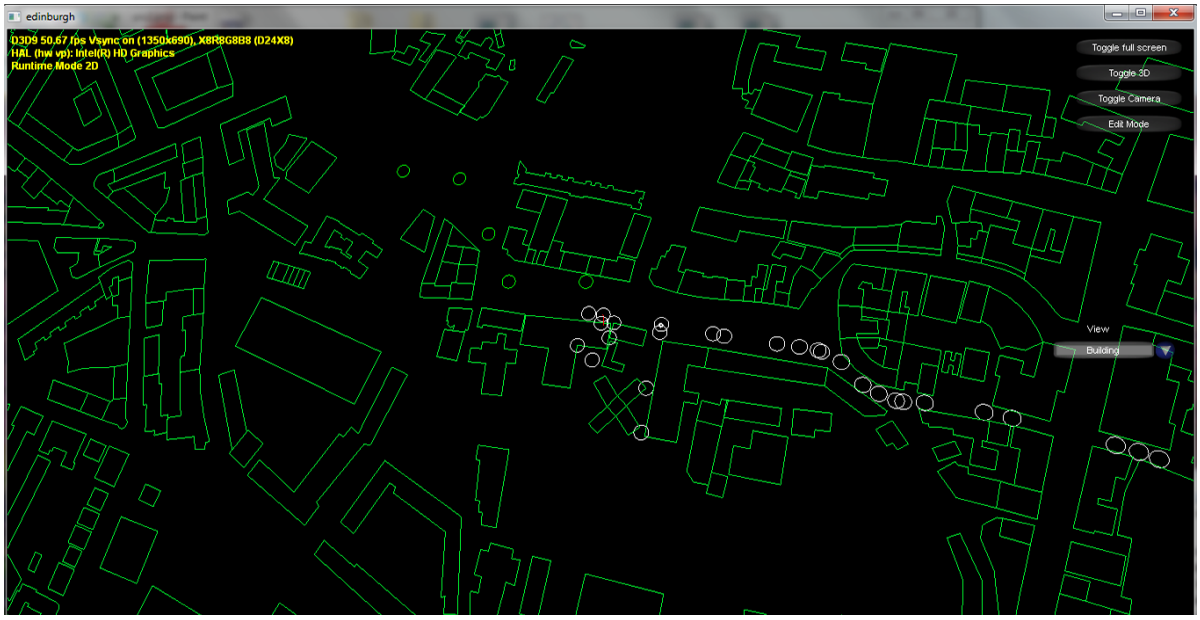

Figure 2: A graphical 2D representation of the Spatial Module's object model.

The simulated user position is updated once every second, the same frequency as the Spacebook phone app would send GPS updates when used by a human. Line of sight queries are shown as red lines while they are calculated.

Only buildings are rendered as a default. However, the rendering of all objects in the spatial module such as roads, areas, amenities, etc can be turned on and of at any time using the drop down list at the bottom right side of the interface.

It's also possible for a human operator to use the GUI to control the movement of the simulated user.By double-clicking somewhere on the map, a new navigational goal for the simulated user is created.

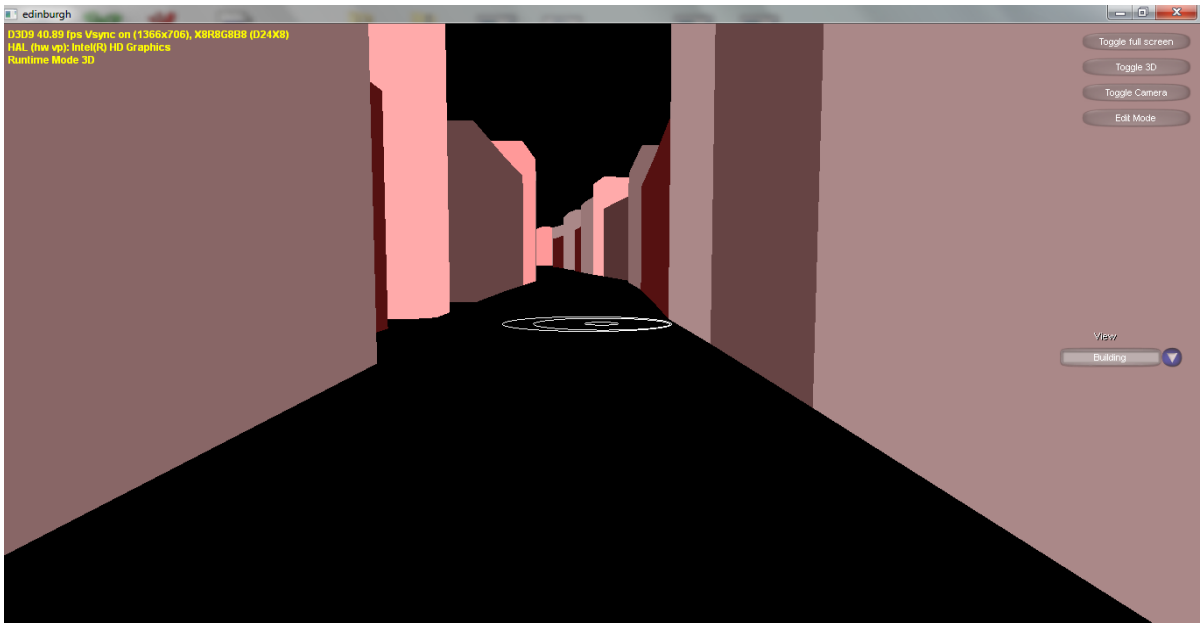

Figure 3: A 3D view of a part of the object model from the simulated users perspective.

A 3D mode can be toggled in which the camera follows the position and rotation of S, showing a 3D rendered version of the environment from the perspective of S.

Figure 4. The GUI in edit mode.

In edit mode it is possible to create any number of triggers by clicking anywhere on the map and attaching an id to the trigger. When S enters a trigger area , an event is sent over ICE to the modules listening for the specified trigger event. This is a very useful feature during development and testing.

There is also a GUI for controlling the simulated user with written natural-language route instructions (see figure 5). The instructions (in this case, "turn left") are entered by the human operator in the top-most textbox. After the instruction is entered, the textbox below shows a paraphrase of the utterance after contextual interpretation (in this case "turn left at 21545086", where 21545086 is the identifier of a node in the geographical model of the city). The utterances of the simulated user are shown in the "Reply" textbox (in this case "where should I go now"). Below the current coordinates and the stacked-up short-term goals are displayed.

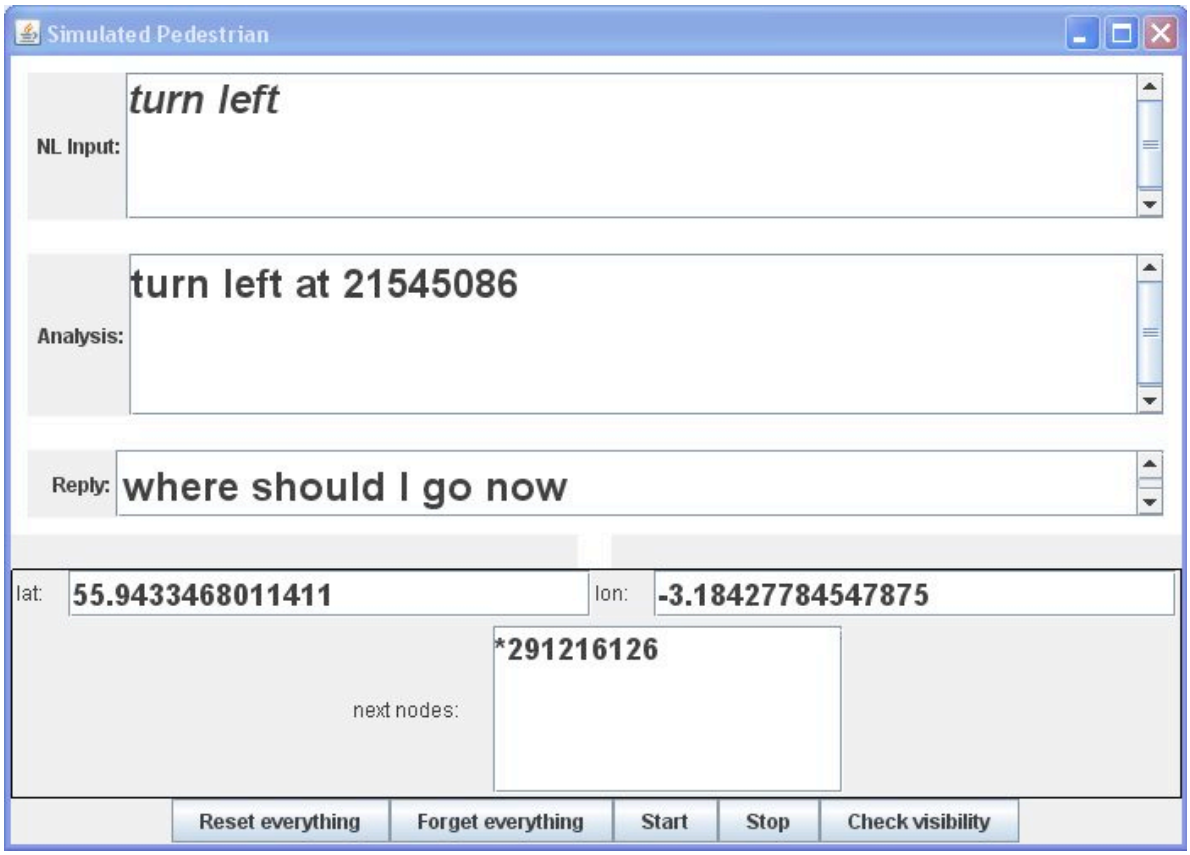

Figure 5. The dialog GUI

7. Visibility calculations

A visibility calculation is triggered by a query from the Cognitive Module, consisting of two lat-long points A and B. The answer returned from the Spatial Module contains the required visibility information (true or false). In the case of "true", the Spatial Module also returns a list of all geographical objects intersected by the line-of-sight between A and B. The answer could for example contain information that there is a road or a park between A and B.

Modeling a city area involves modeling a large number of polygons, and therefore we employ spatial partitioning using a so-called quadtree (using an implementation technique suggested by Prichard, 2001) to increase efficiency.

The top node of the quadtree represents the whole area. The area is then divided in four equally sized quadrants, each of which is represented by a child node. The buildings that are completely contained in a quadrant are associated with the corresponding child node, whereas buildings whose perimeter crosses the border between two or more quadrants are associated with the root node. Each of the four quadrants is then divided into sub-quadrants, and the same process continues recursively a fixed number of steps, or until a sub-quadrant contains no buildings at all (see Figure 6). During runtime, the quadtree allows for quickly finding the closest building of any point in the area.

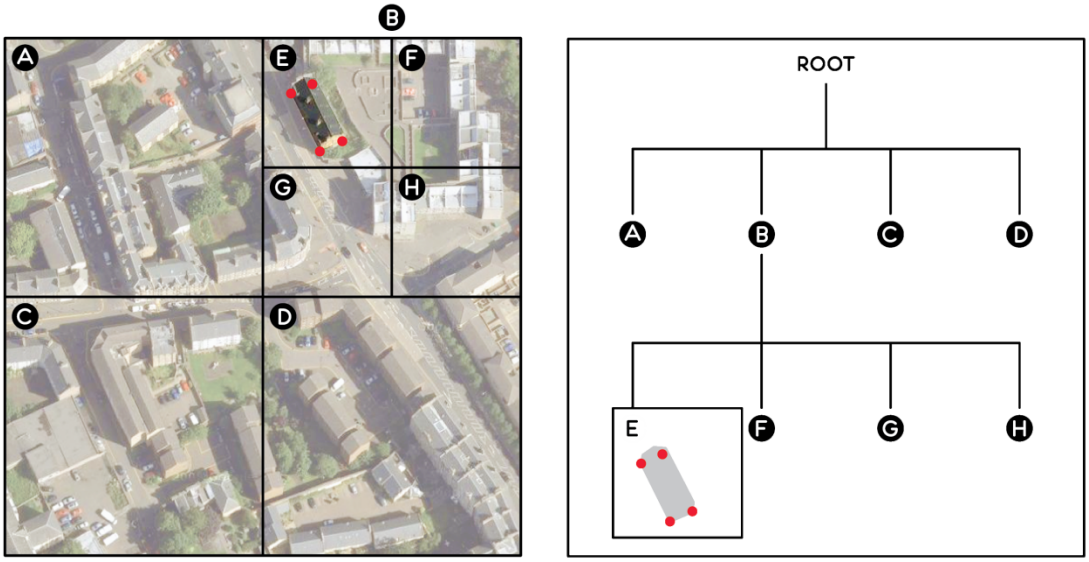

Figure 6. Left: Recursive spatial partitioning of an area in the city in smaller and smaller quadrants. Right: The corresponding quadtree.

An important step in speeding up calculations involving a large number of buildings is to compute a bounding rectangle around each building. The most straightforward choice is the rectangle formed by the lower left point and the upper right point of the building. However, if the building is slanted with the respect to the coordinate system, the bounding rectangle will be unnecessarily large (see Figure 7).

The better approach is to instead make a change of the basis to a coordinate system whose axes are parallel to the perimeters of the building. If this is not possible (e.g. because the building is not rectangular), the bounding rectangle can be made as small as possible using principal component analysis (see e.g. Eberly (2001), section 2.5.2 and Lengyel (2012), section 8.1).

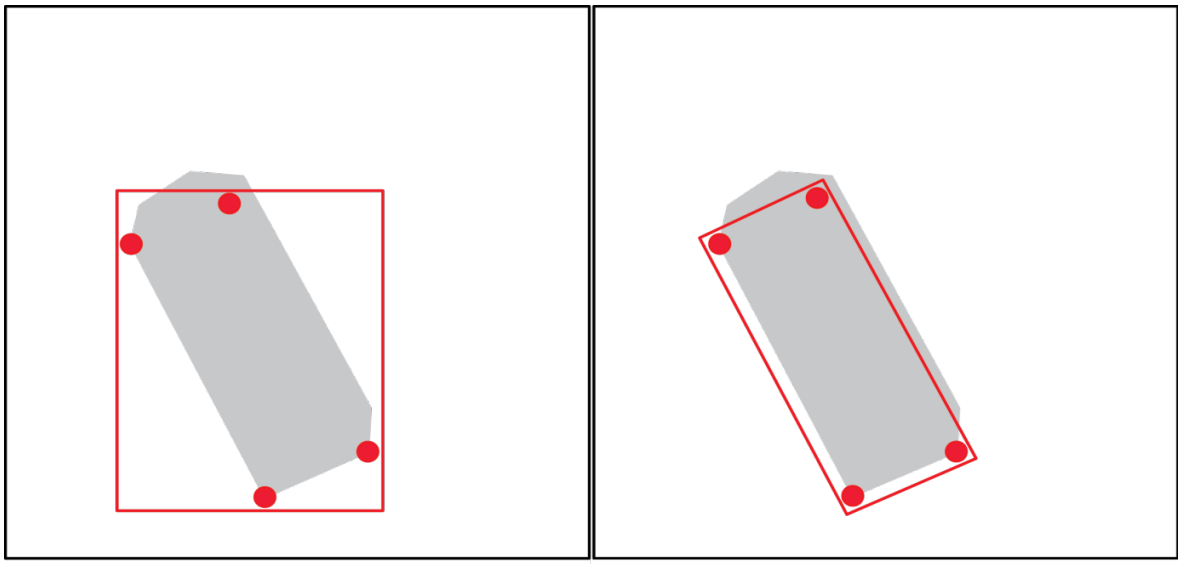

Figure 7. Left: Minimal bounding rectangle aligned with the original coordinate system. Right: Minimal bounding rectangle after a change of bases.

For each line-of-sight query, first a check is performed to see if there are any bounding boxes between the two points A and B. This step allows us to quickly exclude the majority of buildings from further computation. For each bounding box intersected by the line AB, a second more expensive check is performed to see whether AB intersects with the object's polygon. If this second check shows that no building is intersected by AB, then there is clearly a free line of sight from A to B.

Now consider the case where the line AB intersects a polygon which represents a building. It is perfectly possible that B is situated within that very building, but is still visible. This could be the case, for instance, if B represents a shop or a restaurant, which is located within a building but is still visible from the street outside. The heuristic solution adopted is that, for certain types of nodes like restaurants, cafes, pubs, shops, etc., the location of B is projected outside of the building polygon (see Figure 8).

If the point B represents a whole building (e.g. a church), B is usually centrally placed within the polygon, and there is no obvious projection direction. In those cases, B is considered visible from A if the line AB intersects only the building polygon surrounding B. tIf AB intersects a polygon representing a building, and B is not within that polygon, B is not visible from A.

If AB intersects only polygons which are not buildings, B is considered to be visible from A, and the intersecting polygons are returned as a reply to the visibility query along with a "yes" answer.

If there are no buildings between the points A and B point B is returned as visible.

The visibility calculations are performed on the graphical processing unit (GPU), but can also be done on the CPU.

8. Walk-path calculations

To perform walk-path calculations, an array with the resolution of 1*1 meters is superimposed on top of the city map. Walk-path calculations are made with a combination of the A*search algorithm, where each square in the array corresponds to an atomic step, and an influence map representing past behaviours of real users.

Influence maps is a common technique in game programming AI, widely used in real-time strategy games, simulations, first person shooters and similar applications. What these applications have in common are dynamic and highly detailed environments with a large number of active game objects, and the need to produce believable but not overly predictable results fast. Influence maps allow a practical way of letting past events affect the probability of future events in the same location.

In this domain, the influence map is also represented as a 1*1 meter array. Every square of this array is given an initial value of 0.

The influence map is modified by processing logs of walk-paths of real pedestrians as measured by GPS/Glonass that are read into the city model.

if a square in the grid has been trod by a user it's corresponding value in the array is incremented. After a large number of walk-paths have been processed, popular spots will have high values and less popular spots will have lower values. With the exception of GPS errors, inaccessible spots will have the value zero.

There are several ways to create the influence map from user walk paths.

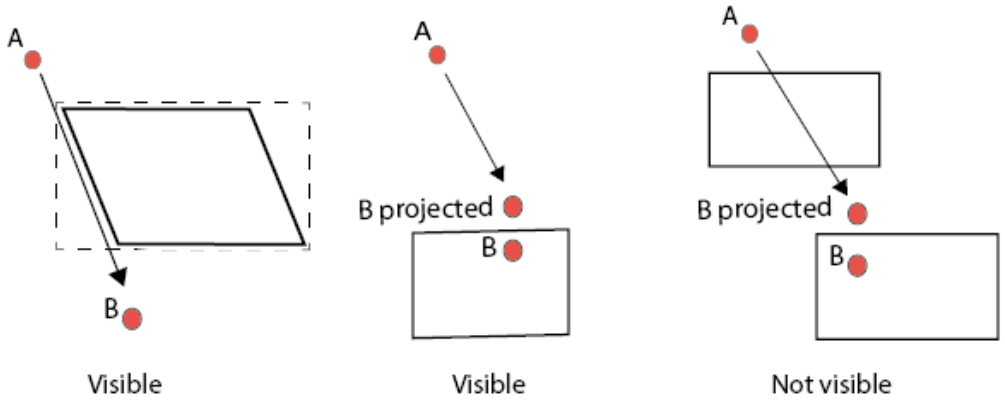

Figure 8: (Left) B is visible even though AB intersects a bounding box. (Middle) B is visible even though it is within a polygon. (Right) B is not visible.

The most straightforward way is to just modify the values of the those array cells where the user has walked. This is illustrated in figure 9a (numerical representation) and figure 9b (graphical illustration). An alternative way is to let also the surrounding squares be influenced in a gradient fashion, This is illustrated in figure 9c (numerical representation) and figure 9d (graphical illustration).

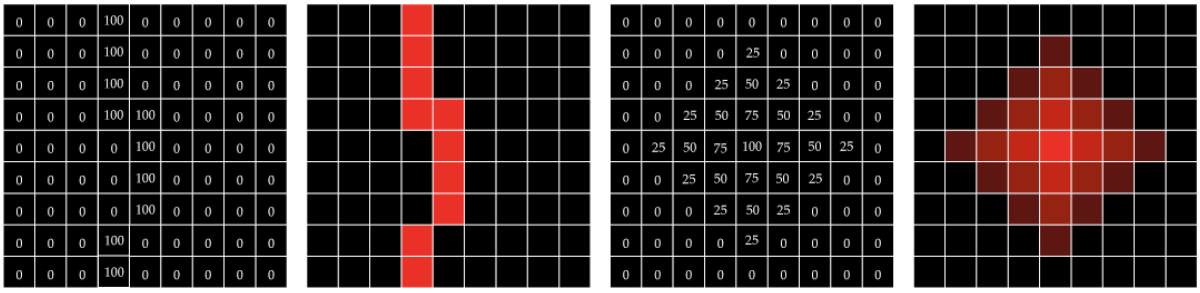

Figure 9. Numerical and graphical representation of influence maps.

Every position along the path of a user is given a positive influence value that can be adjusted by factors such as accuracy values from the GPS and the smoothness of the path. A higher positive value on a square asserts a higher influence on the behaviour of the simulated user.

When constructing a walk-path for the simulated user, the influence map is first added to the 2D grid representing the surrounding area, and the previous user paths are used to lower the cost of each node according to the value in each square . The path is then created using an A* algorithm as described by Russell and Norvig (2003).

Buildings normally have a high enough cost to make A* exclude them from paths. However, if several real users walk through (what seems to be) a building, the cost of walking through the building is lowered to the point to where A* can include it in a walk-path, allowing a simulated user to also walk through. This is very useful since there might be passages which are not evident from the map representation, for instance, what seems to be a building might be an archway.

Whether or not to allow the simulated user behaviours to override map constraints can be set by tweaking the cost of crossing buildings which can be made dependent of the quality of the geographic data available. The simulated user will follow the paths set by real users when available. This also means that if real users avoid an obstacle that is not in the city model, so will the simulated user.

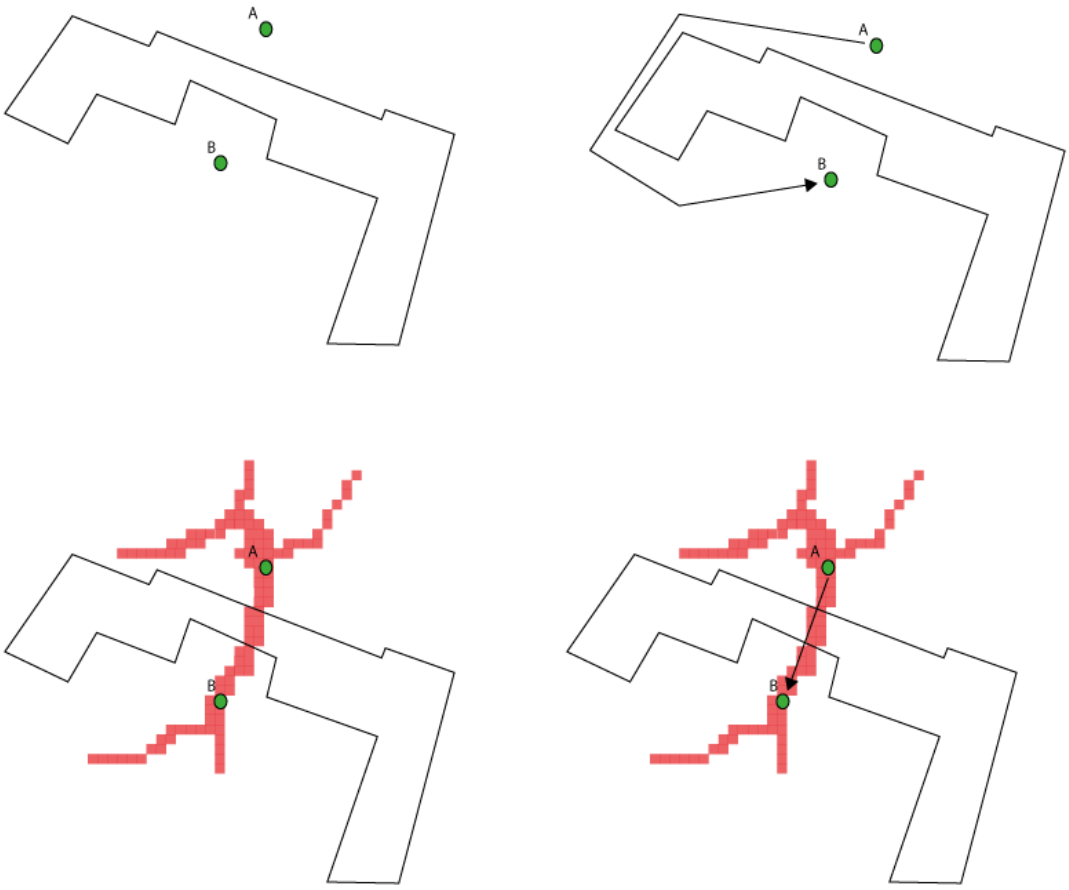

Figure 10. (Top) The planned path from A to B goes around the building. (Bottom) Influence maps built from walk-paths of real users reveal that there is a passage through the building. The path planner uses this information to find a shorter path.

9. Final remarks

Future work includes expanding on the capabilities of the simulated users, and using these users for bug detection and fine tuning in the Spacebook route giving system.

10. References

Albore, A. et al. (2013) Final pedestrian behaviour component. Spacebook deliverable D2.3.2.

Eberly, D. (2001) 3D game engine design – a practical approach to real-time computer graphics. Morgan Kaufmann.

Eckert, W., Levin, E. and Pieraccini, R. (1997) User modeling for spoken dialogue system evaluation. Proc. ASRU'97, pp 80–87.

Götze, J., Scheffler, T., Roller, R. and Reithinger, N. (2010) User simulation for the evaluation of bus information systems. In Spoken Language Technology Workshop (SLT), 2010 IEEE, pages 454–459, IEEE.

H Ai, F Weng - Proceedings of the 9th SIGdial '08 Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue pp 164–171.

Lengyel, E.(2012) Mathematics for 3D game programming and computer graphics, 3rd edition. Course Technologies.

Pritchard, M. (2001) Direct access quadtree lookup. In Game programming gems 2, Charles River Media publishers, pp. 394–401.

Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach, 2nd edition, Prentice Hall.

Schatzmann, J., Georgila, K. and Young, S. (2005) Quantitative evaluation of user simulation techniques for spoken dialogue systems. Proc. SIGDial.

Traum, D. (1999) Speech act for dialogue agents. In Foundations of rational agency, Wooldridge and Rao (Eds), pp. 169–201.