GAMA Model Documentation

Srirama Bhamidipati, Erika S, Arend L

2018-05-31

Contents

Background

It is expected that in 2030 about 5 billion people will live in cities as compared to the 3.6 billion now. This massive growth challenges the liveability of urban environments. Designing sustainable cities attractive to tourists, industry and commercial organizations as well as pleasant to live in, requires a thorough understanding of urban dynamics and the role of individual and collective human decision- making within this.

Figure 1: “Background”

Chapter 1

Introduction

Cities like other complex adaptive systems commonly develop in an open-ended and often unexpected way with neighborhoods and urban populations in constant development. These urban dynamics exhibit many wicked problems in which the issue at hand as well the potential solutions are strongly dependent on the perspectives of the multitude of stakeholders involved.

Conflicting use-patterns among city users impact a multitude of urban quality indicators and result in specific spatial-temporal patterns and conflicts (Fig. 1). Currently, our understanding of these complex processes remains limited and even more so our capacity to anticipate these complex developments.

1.1 Objective

Against this backgound the objective of the City Simulation Lab is:

1) To develop a City Simulator toolbox including various agent-based models of urban systems
2) To increase our current understanding of complex urban processes leading to specific spatial-temporal patterns through simulation of urban systems through the implementation of the agent-based simulation toolbox.
3) To improve the capacity to anticipate complex urban dynamics through interactive participatory workshops developing and implementing a Participatory Urban Modelling (PUM) framework.

1.2 Methods

1.2.1 Case: Tourism expansion in Amsterdam

Amsterdam has seen a steep 40% increase in tourists in just five years, with currently 17 million people visiting the city on annual basis. This strong increase impacts the liveability of the city and causes conflicts among different users of the city.

1.2.2 Participatory Urban Modelling (PUM)

In order to include the different perspectives of local stakeholders, we developed a Participatory Urban Modelling (PUM) framework. Within this cyclic process researchers and stakeholders actively collaborate in the development of the City Simulator toolbox through a combination of (simulation) games (Fig.2). Through interactive gaming workshops, stakeholders collectively define potential policy changes and explore the impacts on quality indicators and spatial conflicts.

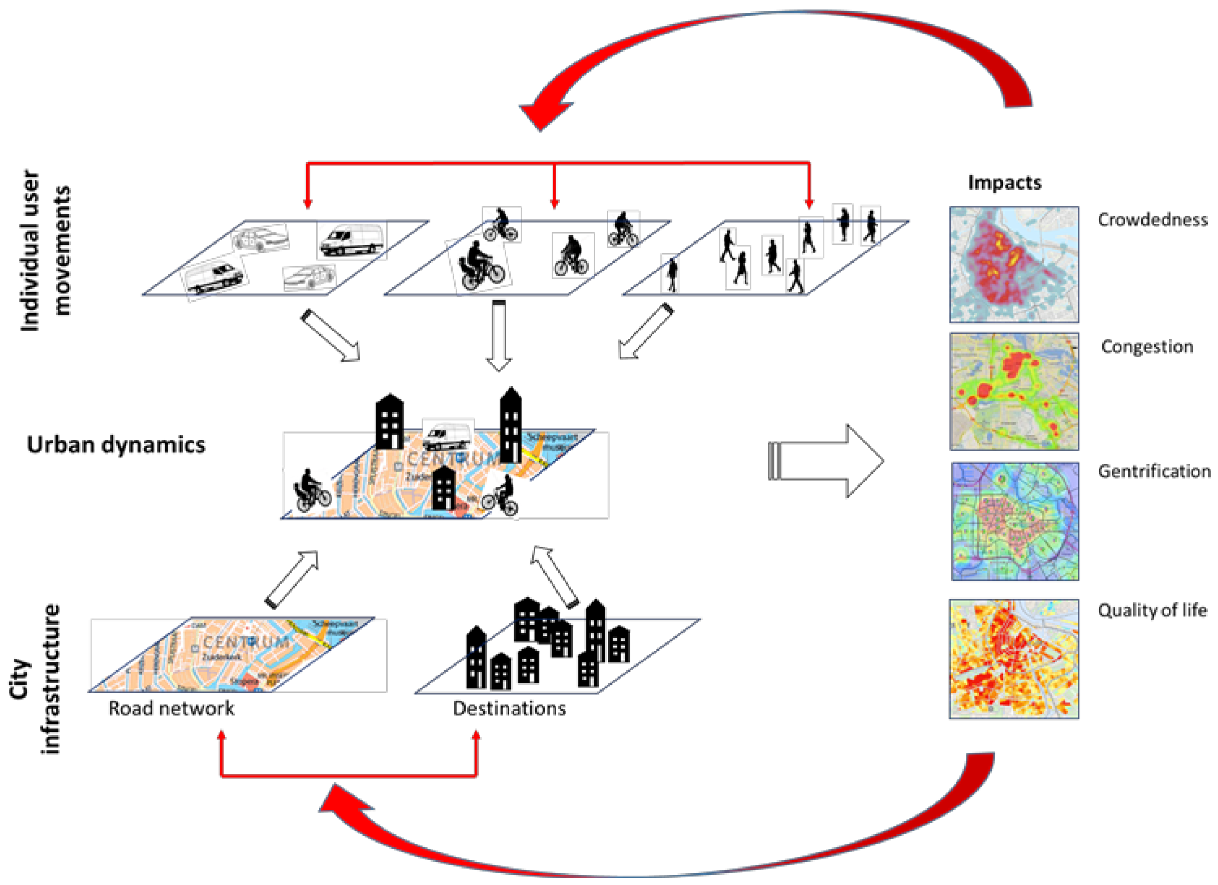

Figure 1.1: “PUM”

Chapter 2

Imports

Chapter 3

Species

The model has 4 species [inhabitants, buildings, roads, public transport system]

3.1 road

This species is imported from a shapefile. The attributes of this shapefile are imported and mapped into various model variables - as attributes of the road species.

3.2 buildings

This species is imported froma shapefile. The attributes of this shapefile are imported and mapped into various model variables - as attributes of the building species. The main classification of buildings is based on its usage. Curerntly the model only distinguishes between [office, home].

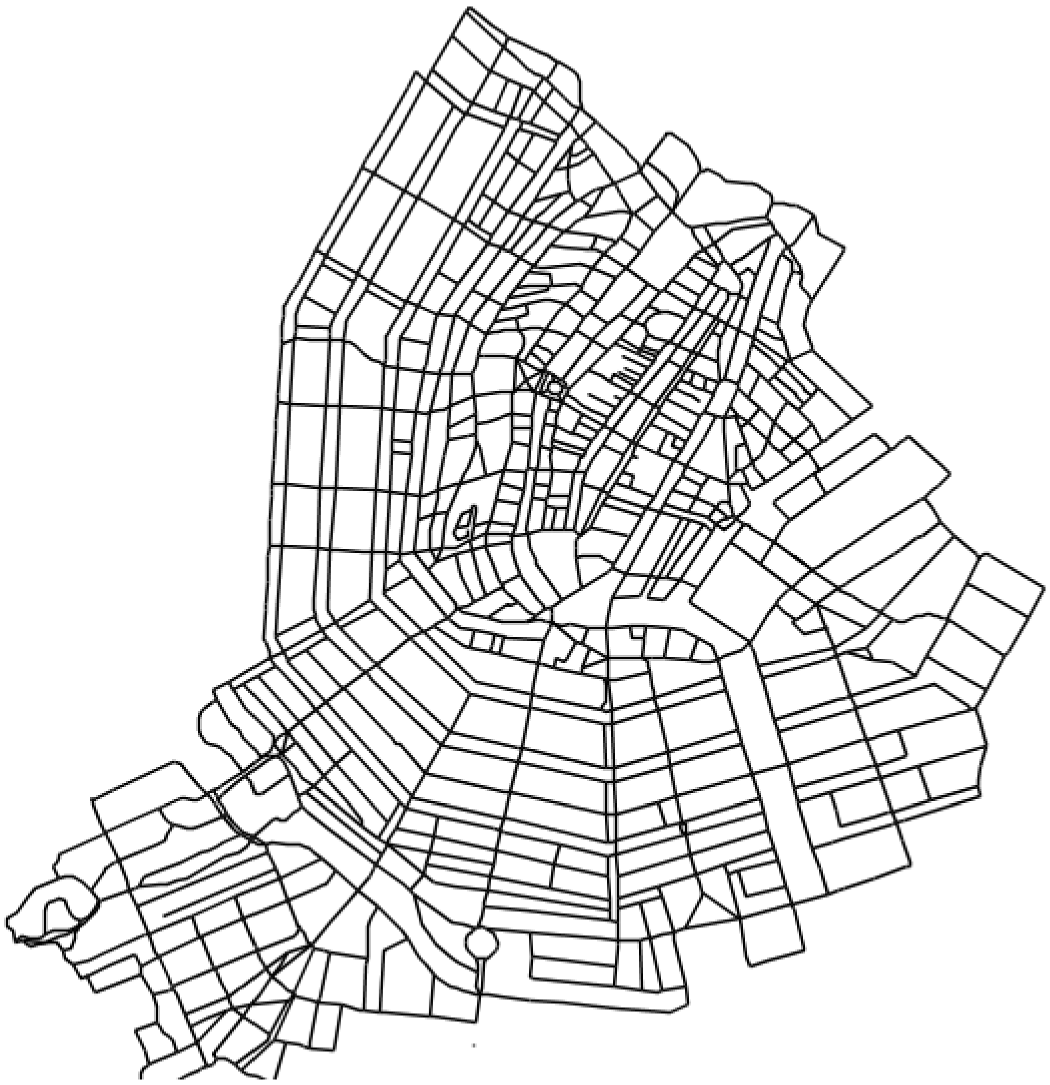

Figure 3.1: “Road Network”

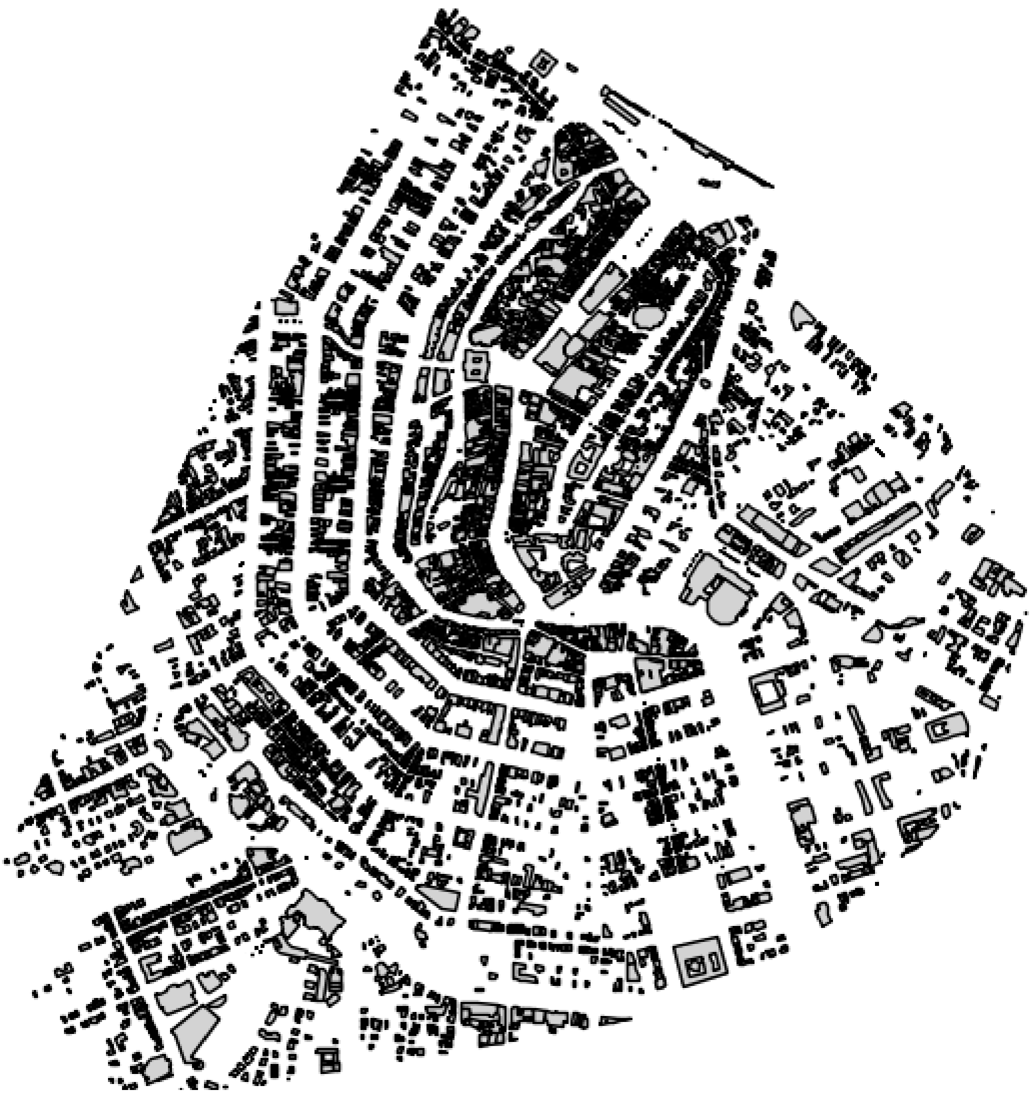

Figure 3.2: “Buildings”

Chapter 4

Functions

4.1 update_mode_specific_memory()

Agent memory on travel time is updated every day based on its experience today. The agent uses last five days of memory to estimate the next days travel time. This memory is mode specific. In this model, agent has 4 memories [walk, bike, pt, car].

```
action update_mode_specific_memory (float tt, int mode)
  {
        //tt is morning_travel_time
        add tt to:self.mode_specific_memory[mode];
        remove index:0 from:self.mode_specific_memory[mode];

    }
```

arguments

- travel time (`float`) : the travel time experienced in the recent trip
- mode (`int`) : the mode used by the agent in recent trip

returns

- empty

usage

- the following example updates memory of mode 1 = walk.

```
do update_mode_specific_memory(travel_time, 1)
```

4.2 evening_movement()

This function makes the agent move from office to home in the evening.

```
action evening_movement
    {
        float my_speed <- mode_speed_int[self.value_mode_actual] ;// # m / # sec;
        do goto target: any_location_in(my_home) on: g speed: my_speed;
        if the_target = location
        {
            my_evening_home_arrive_time <- current_date;
        }

    }
```

arguments

- none

returns

- empty

usage

```

do evening_movement;
```

4.3 morning_movement()

This function makes the agent move from office to home in the evening.

```
action evening_movement
    {
        float my_speed <- mode_speed_int[self.value_mode_actual] ;// # m / # sec;
        do goto target: any_location_in(my_home) on: g speed: my_speed;
        if the_target = location
        {
            my_evening_home_arrive_time <- current_date;
        }

    }
```

arguments

- none

returns

- empty

usage

```
do morning_movement;
```

4.4 execute_a_behavior()

This function executes a behavior depending on the return value from the function *choose_behavior()*.

4.5 choose_behavior()

This function determines which of the four behaviours an agent should follow. This is based on the thresholds for the four quadrants in the consumat format.

```
string choose_behavior(float satisfaction_factor, float uncertainty_factor){

        if (satisfaction_factor > 0.5) and (uncertainty_factor <= 0.5){
            return "repeat";
        } else if (satisfaction_factor > 0.5) and (uncertainty_factor > 0.5){
            return "imitate";
        } else if (satisfaction_factor <= 0.5) and (uncertainty_factor <= 0.5){
            return "optimize";
        }

        return "inquire";
    }
```

arguments

- It takes two arguments: satisfaction (`float`) and unertainty (`float`) as float values.

returns

- It returns a string among [“repeat”, “imitate”, “optimize”, “inquire”].

usage

```
behavior <- choose_behavior(float, float)
```

4.6 get_morning_departure_time()

This function is used to get a departure time for the agent to leave home to office for the purpose of work.

arguments

The function takes no arguments and defaults to 8 am in the morning with a deviation of half hour. The minutes are picked by a random number between 0 and 59.

returns

The function returns a list of 2 integers: the morning_hour and the morning_minute

usage

```
list <- get_morning_departure_time()
```

4.7 get_evening_departure_time()

This function is used to get a departure time for the agent to leave office to home for the purpose of resting.

arguments

The function takes no arguments and defaults to 5 pm in the evening with a deviation of half hour. The minutes are picked by a random number between 0 and 59.

returns

The function returns a list of 2 integers: the evening_hour and the evening_minute

usage

```
list <- get_evening_departure_time()
```

4.8 get_peers

This function is used to get the peers of an individual inhabitant.

arguments

The function takes 3 arguments: `list` of inhabitatnts among from whom the peers should be selected; a constraint on how far (`float`) the inhabitants home should be located from agents owns house; a constraint on how far (`float`) the offices should be from agents own office to consider those inhabitants as peers.

returns

a `list` of inhabitants.

usage

```
do get_peers(list, float , float)
do get_peers(inhabitants, distance, distance)
```

Chapter 5

Reflexes

Chapter 6

Displays

Chapter 7

Experiment

Index