Simulation-Based Generation of 3D Urban Environments using a Multi-Agent System

Reza Haddadi        Andrew Jönsson

June 12, 2009
Abstract

Multi-agent systems is an AI technique used to solve various problems. The intention of this study is to determine how suited this technique is for generating urban environments usable in computer simulations. Other procedural methods usually take an approach in which they statically create a city (e.g. the Grid-Layout method). The multi-agent system used in this project creates a city dynamically on a temporal (simulated year by year) basis.

A prototype tool called Urban, based on a multi-agent system where agents correspond to the real world developers in a city, has been created in order to evaluate the technique. The multi-agent system solves the posed problem by creating a basis for developing detailed and unique cites with different characteristics. However, there is a need for further detailed analysis of the algorithm and of how the extraction of characteristics from the real world actors and environment should be coordinated to maximize the realism of the generated result without making the multi-agent system too complex.

Sammanfattning


## Contents

1 Introduction 3  
  1.1 Background 4  
  1.2 Problem description 4  
  1.3 Structure 5  

2 Domain description 7  
  2.1 Multi-agent systems 7  
  2.1.1 Aspects of using a MAS 8  
  2.2 Modeling of urban environments 9  
  2.2.1 Mimicking real world feedback effects 10  
  2.2.2 Procedural methods 11  
  2.2.3 Data oriented methods 12  
  2.2.4 Semi-procedural tools 13  
  2.3 Previous attempts of using MAS to model cities 13  

3 Design and implementation 14  
  3.1 Data representation 15  
  3.1.1 Land areas 15  
  3.1.2 Roads 17  
  3.1.3 Others elements 17  
  3.1.4 IO tools 18  
  3.2 Agents 18  
  3.3 Terrain generation 20  
  3.4 GUI 21  
  3.4.1 2D rendering 21  
  3.4.2 3D rendering 23  
  3.4.3 Statistics 24  

4 Results and discussion 26  
  4.1 Extendability and adaptability 27  
  4.2 Performance and stability 28  
  4.3 Realism 30
5 Conclusions

5.1 Summary .................................................. 31
5.2 Future work ............................................... 32

A Image results ............................................ 36
Chapter 1

Introduction

Using computers to run simulations is an expanding field both in the industry, when evaluating products, and in the home of everyday people, in the form of computer games.

One of the difficulties of any simulation which is to be used in interaction with humans, is succeeding in mimicking the real world, i.e. creating a realistic environment capable of capturing the attention of the user and convincing him or her to act as if being under the same circumstances in the real world. Even if a perfect environment is manually created it is quite possible that the interacting humans, which are to be tested or trained, have learned how to act in that specific environment and not in the general case. This means that the content creator is forced back to the manual task of creating more environments until it can be empirically shown that the simulation goal has been attained under any circumstance or rather in any city.

In this thesis we try to solve the content creation problem, i.e. how can we remove the need for manual work when creating models for use in artificial environments. In particular we focus on the area of urban environments. A prototype of a tool for creating urban environments, based on a Multi-agent system (MAS), is developed and its qualities compared to other procedural techniques. When creating Urban, which is the working title of the tool, the focus has been on making a tool capable of creating urban environments of varying types thereby satisfying a wide range of needs.

Emphasis when developing the tool has been on enabling it to generate realistic urban environments. When looking at other techniques it is clear that they are primarily bent on populating the city with urban objects and that they pay little or no attention to the actual real-life reasoning done when houses or roads are created, or why they are created. In comparison, in a MAS, where each agent is responsible for the creation of only one type of urban objects, e.g. residential buildings or roads, it should be possible to simulate such reasoning. Furthermore, by controlling the behavior and reasoning of the individual agents it should be possible to passively dictate
the characteristics of the created city but without compromising the desired random layout of city objects.

1.1 Background

Earlier research on procedural modeling of artificial environments has been quite successful. There exists a wide range of techniques that can assist a content creator in the task of modeling buildings, trees and large-scale landscapes [2].

In the case of modeling urban environments research has been made and different techniques tried with varying degrees of success. Amongst these there are techniques such as Grid layout, Template based and L-systems, which are all described in section 2.2.2.

A study of the research projects utilizing these techniques show that they focus on the instantaneous creation of a city with a given size and with urban objects of a fixed architectural level. Most of these have a road approach, in which the city is built by primarily focusing on creating a realistic-looking road network and then creating buildings between the roads [2].

The reasoning behind the above mentioned solutions to the content creation problem is sound in the sense that the city road network layout has an immense impact on everything that happens in the city. Therefore it defines the limitations of the city and on the people living in it. However if the goal is to generate realistic urban environments it is unlikely that an algorithm, in one simple pass, can recreate a city which is in fact the complex result of hundreds, possibly thousands, of years of development and changes based on a multitude of historic, cultural, social, technological and geographical reasons.

1.2 Problem description

The problem with manual content creation is that it requires resources, reducing the available time and effort which could have been spent on improving other parts of the system in question. Since the availability of content for simulators and games is a fundamental cornerstone, the problem can not be ignored if we want the simulation domain, with the possibilities it introduces, to continue growing.

Since the urban environment is the primary domain for humans to live and interact in, finding some way to generate such content becomes very important when the simulation involves interaction with humans.

In this thesis we aim to evaluate the proficiency of a system based on the MAS technique, in solving the content creation problem, i.e. how can 3D cities be built in a procedural and automatic way.
The problem with earlier approaches is that there is no natural correlation between real-world factors and the mechanisms in these procedural techniques that govern the creation of the urban environment. Because of this, there is no natural way of controlling and adapting the production mechanisms of the city. The problem of manual content creation is not solved until a controllable production method exists.

To solve the problem satisfactorily we need to create a fully-automatic 3D urban environment generator capable of creating multiple kinds of cities, where each generated city is unique in layout, without the need to change the generator code. It is also a requirement that the generated result is free from anomalies. This means that no urban object should be placed or constructed in such a way that it is useless to the inhabitants of the city. Examples of such phenomenon could be a highway ending without transition to other roads, roads that are not connected to the main road network or housing areas which can not be reached from other parts of the city.

It is also of importance that the generator is stable, meaning that it will produce, each and every time it is used, a city, in such a way that the generated result adheres to the input data and parameters.

Finally, the system should be computationally efficient and scalable, and it should be possible to use it without the need for a high-end computer.

In summation, the system should:

• support fully automatic procedural generation.
• not produce anomalies.
• use parameters and not explicit data as input, enabling control over and predictability of the result.
• be scalable.
• be usable on a modern personal computer.

Using these requirements as a basis for our evaluation it should be possible to get an estimate of the proficiency of the MAS technique in the specified domain, even though we are limited in the sense that we only have access to a prototype tool.

1.3 Structure

The contents of this report is organized as follows. Section 2 contains background information about the domain, starting with a description of multi-agent systems, both in general terms and what needs to be considered when developing one. The section also describes other previously tested methods for creating urban environments. In section 3 both Urban and the design
choices made when designing *Urban* are described giving the reader a breakdown on the issues which need to be addressed when creating a similar system. Next comes section 4 in which the requirements outlined in section 1.2 above are compared to *Urban* and an analysis of the result is made. Finally, in section 5 a short summary of the important issues discussed in this report is presented. This is followed by conclusions and suggestions for future work.
Chapter 2

Domain description

2.1 Multi-agent systems

Multi-agent systems is the general term for any system, artificial or real, in which multiple agents interact with each other and with the environment. While the agents in a MAS are based on the standard template for agents pictured in figure 2.1, perceive - reason - act, an important characteristic that defines a MAS is that it contains multiple agents who are all autonomous, meaning that each agent has its own goals and its own way of reasoning about the world. This does not however mean that agents are totally independent of each other, since an agent reasons and chooses its next action based on how it perceives the world, or parts thereof, and it is quite possible that earlier actions by other agents may have altered the environment.

![Figure 2.1: The standard agent-environment interaction: Perceive - Reason - Act.](image)

This trait of co-dependence in-between agents and environment might seem unwelcome in a computer program but is in fact useful in cases where it is possible to predict the behavior of single agents but not the result of their interaction. By extracting the characteristics of the real world actors,
using these to create corresponding artificial agents and allowing them to act in an artificial environment it is possible to study and simulate the complex interaction between agents and the resulting system.

Because of the emphasis on studying the interaction in the system over time and not the individual agents the term *Multi-agent simulation* is sometimes used instead of *Multi-agent system*.

### 2.1.1 Aspects of using a MAS

The main advantage with a multi-agent system is that it is possible to emulate a system by describing the agents in it. This means that should it be deemed that the system is not modeled correctly, improvements can be made by manipulating and better describing the reasoning of the individual agents acting in the system.

This feature of emulating system behavior requires however, if any assurances are to be made regarding system stability, that any computer-based MAS needs to deal with the consequences of placing agents in an artificial environment simulated with a limited amount of computational power.

Such problems can involve resource-sharing issues, both inside and outside the artificial environment. Resources inside the environment can for instance be a natural resource system that which limit the capabilities of each agent, e.g. a monetary system. The resources outside the system could for instance be the limited computational power which sets the need for determining run-time allocation for each individual agent or for creating a strategy concerning how agents are run, in what order and for how long.

Another problem involves how conflict-resolution is handled, i.e. what should happen if the action proposed by one agent clashes with the action proposed by another. Typically in the real world this situation rarely happens because humans reason and act in continuous time. If pre-existing conditions suddenly change they have the ability to quickly react to the change. The same can not be said for a computer-based agent which checks conditions of the environment sequentially and makes decisions on the perceived state of the world. If a condition which one agent is very reliant on, for some reason rapidly changes value, we run the risk of the agent acting on the perceived and not the actual state of the environment, thereby possibly putting the environment data in an erroneous state and destabilizing the system.

It is also possible for the simulation to experience *starvation* issues, if agents have to compete against each other for computing time, e.g. when the time it takes for agent to perceive-reason-act is dependent on how the agent is coded, and the system does not adjust for this inequality between agents.

To avoid these issues strategies that govern the acting of agents can be employed. Such strategies could for instance be:
Each agent proposes one action. All non-conflicting actions are permitted. If there are conflicting actions, a priority list of the agents determine which agent’s action is performed. Agents whose actions are denied may come up with an alternative non-conflicting action.

Agents are run in a round-robin strategy, as depicted in figure 2.2, thereby eliminating the risk of starvation but introducing a sort of priority list and dependence in-between agents.

Agents are allowed a specific amount of run-time, or amount of decisions. Resources that the agent is about to change are locked from other agents.

All agents propose actions but in case of a conflict the action which maximizes a given global value function is chosen.

In the case of a conflict a routine for handling it may be used. An example of such a routine is an auction in which agents use previously allocated resources to bid on whatever resource they are fighting for.

![Figure 2.2: The round-robin strategy. Only one agent at a time is allowed to act in the environment.](image)

### 2.2 Modeling of urban environments

Because of the steady increase in computing power, generating environments for usage in computer simulators is still being expanded on as the possibilities grow to utilize more complex techniques, which pay even more attention to details. Also important to note is that not only is the content generation faster but the access to cheap memory has also had an effect on the size of the generated content, enabling the creation of greater landscapes, bigger cities and more detailed worlds.
In the domain of urban environments both procedural techniques and data oriented approaches have been tested to various degrees. The former group contains but is not limited to techniques such as Grid Layout, L-systems and Template based methods. What these three techniques have in common is that they create roads based only on trying to achieve a reasonably realistic road network. The areas between the roads are then allocated almost as an after-thought to whatever land zoning type the technique deems appropriate, e.g. residential or industrial.

2.2.1 Mimicking real world feedback effects

In a real city the relationship between cause and effect, in the domain of road construction, is bidirectional. This means that it is both possible for a certain type of land area to be placed based on the availability of transportation in that area, and the reverse relationship, i.e. where roads are constructed in order to satisfy a need for better transportational options. This type of bidirectional dependence and feedback between different urban objects, e.g. roads and housing areas, illustrates why a MAS might be a suitable technique for creating realistic urban environments.

However, a proper characterization of this feedback introduces its own set of problems. The main problem being that of keeping track of what land is currently allocated for land areas or for roads, and in particular how the reallocation of land areas to other urban objects is going to work. This problem is not easily solved when describing a continuous environment, with a limited ability to store and process data and a need to keep the data simple enough for the agents to analyze and work with.

The problems with partitioning and reallocation in 2D space makes the selection of which algorithms to use and the design of data structures a particularly important issue. If we want efficient searching and sorting of the options available to us for choosing, placing and repositioning urban objects when we are modifying our environment, special care must be taken to ensure that we do not introduce anomalies into our result. What is essentially nothing more than looking on a map and locating free and suitable terrain, for humans in the real world, is in the artificial copy one of the main problems which must be overcome if a proficient generator is to be created.

While there are quite a number of things that make the search for tools and techniques which can solve the content creation problem complex, the problem has also some aspects which can be utilized and exploited by a solution. Specifically this deals with the fact that there are multiple levels of detail in an urban environment and when trying to address the problems mentioned earlier we need not always consider the implications on each level. As an example of this consider the case when a land area is to be divided into lots. This is of little concern to adjacent roads but there might be implications that affect the adjacent areas, e.g. how high can buildings be
without appearing out of place. Properly identifying these dependencies and especially identifying the case when there are no dependencies allows the usage of different techniques in such contained cases.

2.2.2 Procedural methods

**Grid Layout** is the most basic of the procedural techniques based on the reasoning that modern big cities are created in a grid layout fashion. In this grid the edges form the roads and the areas they enclose are allotted to buildings. In [1] Greuter et al. present the technique together with a building appearance generation methodology based on geometric primitives. According to [2] the grid layout technique has later been expanded to enable multiple cells in the grid to be used for one building area, forming non-quadratic construction spaces.

While the grid layout technique solves problems with procedural city modeling such as area allocation and basically removes the need for advanced road construction strategies, it does so by reducing the number of different types of cities that can be created. While this type of grid-based road networks appears in cities of all sizes it is, as mentioned above, a road network solution mostly used in larger cities where the need for great coordination and a sustainable road network strategy is of importance. Even more important to note is that even in this type of cities other road patterns are used in combination with the grid layout indicating that a pure grid based road network does not have the potency to create realistic enough urban environments.

**Lindenmayer-systems**, commonly referred to as L-systems, is a technique which has been applied to different problems in the domain of urban environment generation like road and building construction, with satisfactory results. In [5] the authors apply it twice, firstly to the generation of complex building structures and secondly, and from our perspective more interestingly, to the procedural generation of road networks.

The L-system is a technique based on applying rules to entities, e.g. letters, where the rules usually dictate how one entity is expanded into a group of new entities. After the entire line has been parsed and transformed into a new line of entities the rules are applied again and again, thereby expanding the system in each iteration.

Because of the freedom in designing the rules and the possibility of manipulating the entities and choosing how they should be interpreted, L-systems can be used in a wide range of areas. In the case of applying the technique to procedural urban modeling the result, as described in [5], is capable of creating road networks, with multiple levels, for a given terrain and population density map. Thus satisfying both the local need for areas to be connected to the road network, through the creation of streets, and the global need for
arterial functioning roads in the city, in the form of high-ways.

Template based generation is a technique based on road pattern templates, introduced in [6]. While the technique is procedural in the sense that it uses rules when applying the chosen road network pattern and adapts this pattern to the specific map location, all the different templates are dependent on access to population density data for the 2D data grid which is being populated with roads.

In the case of the population-based template, the grid is subdivided into cells using Voronoi tessellation, until the population of each cell is below a given threshold, while in other templates, e.g. the raster template, the population density data only impacts the density of the nodes in the created raster. These two quite different ways of using the data has however the same effect and is based on the same reasoning, as explained in the article. The greater the population density, the greater the need for efficient transportation facilities becomes, i.e. more roads are needed compared to a less populated area [6].

The technique also has other data dependencies; land and water boundary data, in order to know where road construction is possible, and data about the elevation at each point in the 2D space being populated. Using this data each of the roads is examined for validity and possibly modified to adhere to the boundary constraints set by rules of the specific template.

What is important to note is that the two formerly described techniques’ need for population density data suggests that the procedural traits of the techniques are static in the sense that the end result is fixed for any given set of input data. In comparison with a MAS used for urban modeling, neither technique is appropriate for modeling the feedback effects found under real world circumstances.

2.2.3 Data oriented methods

Data oriented methods are based on using data about existing cities and trying to transfer that data, e.g. road and building placement, to an artificial copy. Such data could be detailed height information about urban objects enabling the creation of similar-sized building in the artificial environment. Sometimes this is also used in combination with photo material collected at the same time enabling the buildings to be textured with their actual appearances. This enables the environment to be very realistic when it comes to the size and positioning of the urban objects although it puts an upper boundary on the level of detail of the graphics.

Just like with other techniques used in this domain, it is possible to use the data oriented methods on multiple levels or in combination with other methods. It is e.g. possible to use data obtained from a real city area,
regarding the positioning and layout of the buildings in that area, when
the objective is to populate an area created with a procedural technique.
This type of mixing techniques on different levels enable the end-result to
utilize the strengths of the different techniques while hopefully avoiding their
weaknesses.

2.2.4 Semi-procedural tools

In addition to the previously mentioned methods there exists semi-procedural
tools, for instance Citygen presented in [3], in which the content creator uses
a point-and-click tool to place and customize the fundamentally important
city structures e.g. the arterial roads in the city road network, moving
the nodes around to whatever position is deemed suitable. The tool then
populates the areas which are inside a loop formed by multiple primary roads
with secondary roads and buildings. This type of interactive tools give the
content creator a mix between controlling the layout of the city but with
benefits of procedural creation of the urban objects. It does however not
comply with the stated goal of fully automatic generation.

2.3 Previous attempts of using MAS to model cities

At least one other attempt has been made in which a MAS is utilized for
the purpose of creating urban environments. In [4] Lechner et al. propose
the usage of a MAS in order to “...generate artificial cities that are con-
vincing and plausible by capturing developmental behaviour”. Furthermore
the authors emphasize the MAS technique’s ability to create cities that are
plausible during all stages of the development process.

Lechner et al. recognize the need for a range of different and specialized
agents but have at the publication date of their paper implemented only four
agents. Of these, two are in charge of road construction and another two act
as land developers for residential and commercial zones, respectively.

The tool created by the authors work on an environment made out of
a 2D rectangular grid of land-patches, where each patch can be either of
commercial type, residential type or represent a road segment. Agents are
basically free-roaming entities following existing roads or close to existing
roads. When an agent lands on a patch, it evaluates the consequence of the
different available actions and performs the action which will increase the
value of the patch the most. Should no action be beneficiary to the patch,
no action will be taken.
Chapter 3
Design and implementation

The main goal of this thesis is to evaluate how a MAS can be utilized to procedurally model 3D urban environments. To accomplish this goal a prototype tool is developed, which is based on a MAS in which the different agents mimic the real world developers. Because of the limited time available for development and evaluation some aspects of the content creation problem are examined more closely than others.

As the main task is on evaluating the technique and not the tool, the requirements set down in section 1.2 are all handled to various degrees. The limitations instead concern the level of detail in two different areas of the tool. Firstly, to what degree of accuracy each agent is modeled and secondly, which content in the urban environment do we attempt to model using agents.

Unfortunately, it is not possible to simply skip some parts of the content creation as removing urban objects would eliminate the possibility to evaluate the created cities visually, which is an important aspect of the verification process.

The following limitations were set down before and during the development phase of Urban:

- Buildings are only modeled to correlate with the simulation data concerning height and occupied area.

- Only agents which are very prominent actors in the development of a city are modeled.

- Only a basic terrain generator is created although the option to import terrain data from other sources exist.

- Exporting the generated results is only available for the simple-ASCII format.

As stated above the visual verification of a city is a very important part of evaluating the proficiency of Urban. Because of this, extra requirements
exist with regards to graphical user interface (GUI).

The requirements for the GUI are:

- Be able to render the city landscape in 3D.
- Be able to show a informative overview of the city landscape in 2D.
- Have triggers to start, pause and delay the simulation.
- Have triggers of some kind to change the input values for the simulation.
- Present both local and global statistics about the city for analysis and evaluation purposes.

Even with the template created by the requirements set down on the solution and the graphical aspects of the tool, evaluating how realistic a generated city is, is hard. The discussion and evaluation of the MAS technique is therefore also on the difficulties experienced during development.

3.1 Data representation

Data representation in this project was a continuous struggle since so much data needs to be considered for each decision to be calculated, therefore the best way of memorizing the data was sought.

The original plan was to make the whole world a grid map constituting of one by one meter fractions, where each fraction would contain all the information regarding that square. As soon as the implementation began it became clear that this strategy would not work. Firstly it would require too much memory for any practical use today. A 10 by 10 kilometer grid would require 4000 megabyte of memory if every square would approximately require 40 bytes \( (10000^2 \times 40/10^6 = 4000) \). Searching through this spatially organized data for specific objects would also be very complex.

To solve the searching and reasoning problems we originally planned to have a data structure working in parallel with the grid which listed those land pieces which were in use. Later on this became the primary method for storing and working with objects.

3.1.1 Land areas

ProjectedLand

When it comes to land projection for the different agents the first step is to determine what piece of land will be used. The chosen land would have
to be able to have neighboring lands, roads connected to it and containing areas which are easy to work with in order to actually build anything upon them.

*ProjectedLand* is the name of the class which satisfies these needs. It uses a set of *AdjacencyMarker* to keep track of its edges. These edges can be used find suitable positions for other ProjectedLands. AdjacencyMarkers are also the objects that the road network uses to determine were to build roads on the edges of ProjectedLands.

Within the borders of the AdjacencyMarkers a *QuadArea* will be established. The QuadArea is an extension of the existing Polygon class in Java. It allows the classes working with buildings, which are described below, to calculate smaller QuadAreas inside the main QuadArea.

It will finally be the *Subdivider* that will use these characteristics of the QuadArea in order to proportionally divide the ProjectedLand into suitable spaces for the next step of adding buildings.

In figure 3.1 below a basic UML diagram over the ProjectedLand hierarchy is shown.

![UML diagram](image)

Figure 3.1: UML diagram for the representation of a projected land area together with its most important associated classes.

**Buildings**

Before buildings are built they need a designated area to build on. These are called *Buildspaces*. Buildspaces divide the ProjectedLand into smaller areas which are easier to work with than working with the entire QuadArea of the ProjectedLand. Also BuildSpaces provide a suitable basis for where the building is allowed to be built.
Each ProjectedLand has a unique BuildingPattern that determines where on the Buildspace the building should be located. The BuildingPattern randomly gives the ProjectedLand a pattern describing how the buildings should be placed relating to each other on the Buildspaces. The purpose of this is to decrease the level of monotony and give a more dynamic and unique appearance to each land area.

Finally, there is the Building class which is actually the only thing that can be visually observed. The Building class contains all the relevant information that the agents need for reasoning.

3.1.2 Roads

Roads have a big influence on the development of the city. Because of this there is a need for a road network which looks realistic and connects properly. The RoadNetwork is the class which keeps track of all the roads. Other actors of the city will report their need for road connections to the RoadNetwork. It is then the RoadNetwork's task to see how to best connect two points in world with road segments.

Upon the request to build roads the RoadNetwork will turn to its list of RoadNodes to look for suitable connection points. This can be a complex search so optimization has been made to start looking for suitable junctions close by.

When and if a suitable junction is found then the RoadNetwork knows where to connect. But if the distance to the connection point is beyond the length of one straight road segment it is necessary to build several connected road segments. This will be done because of two reasons. Firstly because it is more suitable to have a smooth road without sharp turns and secondly because these roads will be incentive markers for the ProjectedLands to be built next to. And it is usually preferable to be placed next to a short road than a long one. In some special cases roads can be split up into two roads just to solve the problem of too long roads.

When all the calculations are made the actual Road(s) will be created. The roads are very limited in their utility. They know only to which nodes they are connected. But they also carry the necessary information about their width. The width can indicate if the road is a highway or a narrow downtown street.

3.1.3 Others elements

A great influence on a city's development is its surroundings. In Urban's case it is specifically its neighboring cities. Neighboring cities will, depending on the size, create such attraction towards each other so that roads will be built between them. This will add to the infrastructure of the city being generated, as it is in between, and add to its characteristics.
Also neighbors will, depending on the wealth of the city, help increase growth rate. Immigration will occur depending on the state of the city and its neighboring cities sizes.

\textit{DataGrid} is another a class designed to simplify the work done by the LandProjector when it is searching for suitable land areas. The DataGrid contains a grid of values showing how attractive land areas are. Because searching and grading different options is more common that actually creating the land area it is useful to calculate this value in a a static way instead of calculating it every time the two options need to be compared.

Finally, a class has been made called \textit{Values} where all interesting variables are set to a initial value. These are for instance, average life time of a building and minimum number of neighbors. These values have been collected to this specific place because of the ease to change and observe them.

3.1.4 IO tools

The IO tool has been added as a complement to \textit{Urban}'s own landscape generation as it is not very advanced. The input option enables the use of pre-calculated landscapes. The data that is read is just a single height map using simple ASCII encoding, which is a common format used in geographical information systems (GIS). The height map is used in a very simple way. It will only use height exactly as it is set out without considering where water or trees could appear. Water will only appear when a land is below water level that is defined as zero.

The output option enables the user to save the current setup of the world. The user will get three files; one containing the height map, one containing the roads details and finally one that contains the building data.

The IO tool is not made for extensive use but rather as a way to get some base coordinates to work with.

3.2 Agents

The agent system uses the round robin approach to determine in which order the agent will be working. The agent with the highest importance to the city will run first followed by others with decreasing importance, e.g the \textit{LandmarkAgent} would have higher priority than the \textit{IndustrialAgent}. This priority list can roughly be related to how high value of land each agent aims for. As an example of this; it is pretty common that industry would be found in the outskirts of a city, on cheap land or at least what used to be cheap land. While landmarks of different kinds are much more frequently found in the central, expensive parts of town. Landmarks are here considered to
be cultural buildings as various religious buildings or other buildings of high importance.

What is common for all agents is that they will always start off by calculating the need of the urban objects they are responsible for. The need for increasing amounts of buildings are usually dependent on the size of the city. In some cases they are also dependent on the input parameters that are set prior to the simulation start. One of these is the landmark parameter that will determine the amount of landmarks needed in relation to population.

Next stop is to figure out whether the need is large enough for the agent to proceed or not. If the need is not high enough the agent will abort further work and wait till next year for the same procedure. If instead the need is large enough the agent will proceed to the next step of reasoning.

The agent will fill the calculated need by building new structures to support the increased demand. First it has to collect the land needed. The land projecting part of the program is a quite important one. When an actor of the community demands a piece of land for a certain intention many things have to be considered. Most of these relate to the current state of the world. There are many correlations to where things are placed. This would be very hard to calculate if there would be correlation between each possible new land area and each currently existing land area. The complexity would increase exponentially and therefore it is important to find smart suitable substitutes to this algorithm.

This is where the LandProjector and DataGrid comes in. The LandProjector is the class that provides the different agents with pieces of land. When an agent asks for a piece of land it has plenty of options on what characteristics the land should have. And the LandProjector will do its best to try to provide the desired land to the agent. Among these options there are the value of the land, the availability of major roads and preferred distance to center. These are the dominant options that will decide which land area will be offered to the agent. In table 3.1 below, the land attribute preferences for different agents are listed.

<table>
<thead>
<tr>
<th>Distance need</th>
<th>Transport need</th>
<th>Value need</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residential agent</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>Commercial agent</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>Industrial agent</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 3.1: Dependencies between the three main agents and the data in the DataGrid.

In order to keep the calculations at a minimum, values in the DataGrid are only changed when definite changes to the world occurs. Even then only a small area around the land in question will be influenced.

When the agent has decided what type of land to build and received the
desired land from the LandProjector it is time for the last step. This consist of actually adding the objects that the agent is responsible for. In order to match the demand as many buildings as needed are built. The need is recalculated after each building is built. If the land that was received did not provide for all the buildings needed more land will be requested and the process will continue.

3.3 Terrain generation

Realistic landscapes is the foundation of creating a realistic city. But since the focus of the project is on generating realistic cities and not landscapes it was decided not to spend too much time on landscape generation.

As a complement a function to add pre-calculated landscapes is implemented.

The initial thought was to generate a grid of heights as the height map. This would have to work with the idea of having the landscape variable in sense of changeable parameters. The parameters which were chosen are described below.

**Forest** Which determines the amount of forests and their sizes.

**Mountain** Which determines the amount of hills and their amplitudes.

**Water** Which determines the amount of lakes and their sizes.

Since not much effort would be spent on the generation we needed these parameters to render the grid in a simple way. A simple method to set out all these areas of interest would be to randomly chose locations for them on the map. The amount and size of these fields would vary depending on the value of the parameter.

From here on all we need to do is to calculate every point in regards to the already determined points of interest. A simple interpolation was used to satisfy this need. Basically the mountains would pull other positions up relatively much depending of its distance to the mountain center. Lakes would do the same but pull down and instead with a much faster function, i.e. use a exponential function so that points far off from the center would hardly be affected.

As for forest these would not influence the height grid but instead have its own grid with just the information if there was a tree there or not.

As the project carried on and became bigger and more resource demanding, the need to restrict the data use emerged so an alternative solution was invented. The new option was to dynamically calculate the height when needed. This was fairly easy since the height was just an continues interpolation function.
The option of having coast and rivers flowing through the city had always been an interesting idea since so many cities do actually evolve around them. These options were added by extending the already available lake creating method. Basing our extended method on the fact that lakes and rivers can be described as elongated bodies of water we allowed the method to be more dependent on one of the grid axes. This has the effect of elongating the body of water in either the x or y axis creating a river or a coastline.

3.4 GUI

The GUI was created so that the user can easily setup the parameters and get to see the visualization of the simulation straight away. The tabs will swap to the next tab as one is done with the settings on the previous tab. It starts with the landscape settings and then the simulation settings and finally one arrives at the feedback and statistics tab. In figure 3.2 one can see the GUI in its initial state.

Initially the view starts on 2D which is more useful for overview and feedback on what is going on in the city. The 2D view is color-oriented so that the user will instantly know what type of buildings is being produced.

The user may switch to 3D view for more realistic feedback. The 3D view is equipped with many help tools that can ease the navigation in the world.

3.4.1 2D rendering

An ordinary Canvas from Java’s standard library was used for the 2D rendering. The 2D rendering shows a simple overview of the landscape. The canvas is useful since it can draw simple geometric figures like polygons and pixels.

The height map is drawn pixel by pixel to be able to show height difference in a varying shade of brown. Lakes are drawn blue and trees green. Then on top of this landscape the projected lands and the buildings will be drawn as polygons. These are drawn according to their representing colors. As can be seen in figure A.2, in appendix, green represents residential, blue commercial, yellow industrial and finally pink represents public areas.

To decrease the waiting time for the 2D rendering to redraw each time a change occurs different strategies were implemented in the rendering logics. For instance, new land areas that are to be drawn will just be drawn on to the a copy of the current image and then this altered copy will be posted as the current image to be shown.

Also after each rendering a copy of the currently shown image will be saved so no re-rendering will be needed when switching between the different views.
Figure 3.2: The GUI in its initial state showing the empty canvas and the terrain tab.
3.4.2 3D rendering

The chosen language for the MAS implementation had been Java so naturally Java 3D was chosen to be the choice of 3D library.

Simplicity was the key to create a fast and easy world with low requirements. There were option to use pre-rendered models and texture but that would have required too much processing and memory for a large city, so instead all models are made out of simple shapes.

To add more realism for the visual evaluation simple textures were added. A simple randomizing algorithm was added to avoid repetitiveness. There are textures for walls and roads of different kinds. Also some special texture were added for squares and markets so these would be observable.

![Figure 3.3: The GUI in first person view, somewhere on the outskirts of a city.](image)

Extra effort had to be put on developing tools for viewing the 3D-model. Mouse navigation is very easy and useful, but in a 3D world it can get confusing. So easy keyboard shortcuts were added to view the world from different directions.

There is also the first person view mode as can be seen in figure 3.3. To see the world from different overview angles could probably be sufficient but to see the world from a first person perspective gives a greater depth of realism that can be useful in evaluation of this kind of project.

In order to be able to get to a certain place, in first person mode, a red sphere was placed where were the first person view is located. This will help avoid long direction and orienting problems when the user is trying to find a certain position in the world.
One of the first implemented utilities to analyze and develop was the grid function that is also triggered by a key press. This feature would make the height grid that acts like the surface landscape to become semi-transparent so only the nodes and edges would be visible. And since the grid is made out of 50 by 50 meter squares it becomes easy to analyze and examine the simulation process by measuring how big or small certain projections were. A city rendered in this mode is shown in figure 3.4.

![Figure 3.4: 3D-view of small city. Land texturing inactivated showing the size of each cell with height data in the terrain.](image)

Trees were being rendered in the beginning. This led to great problems in memory consumption. So rendering trees is now an option if the user wants to have a slightly more realistic and dynamic landscape to view. The downside is that one can not render as big landscapes as before. Firstly there will be forest areas randomly located on the landscape. How many forest locations and how big they are depend on their input value. But also whenever residential buildings are created there will be trees planted randomly in their respective yard. This is not very scientific but it adds to a nicer more realistic look as can be seen in figure A.5.

### 3.4.3 Statistics

The statistics tab is divided into three independent fields of feedback as seen in figure 3.5.

The statistics fields gives us raw statistics about the world, such as the current year, population, number of projected land and so on. Using this data the user can track the city creation process and if he or she is interested
in having a city meeting some specific criteria, he or she can read from the statistics field and stop the simulation when the sought after criteria are met.

The grabber tools main purpose is to add as a debugger tool for the developers. It is connected to the mouse cursor in 2D mode so that whatever the cursor points at will be presented statistically in the grabber tool field.

![Figure 3.5: The statistics tab. Grabber tool is showing information for an object.](image)

The simulation log is a log of every action that is performed during the simulation process. There are different categories for what the user or developer can read about. These categories can be unselected and re-selected in order to easier find what one is looking for.
Chapter 4

Results and discussion

In this report we have described the tool *Urban* and the reasoning behind its implementation. As shown by the previously presented images in this report and the images in appendix A, *Urban* has the ability to generate 3D urban environments with alternating layout without the need for interactive user input.

An important thing to keep in mind is that the comparison between *Urban* and other techniques is very hard to perform because of multiple reasons. One of these reasons originates in the fact that different techniques have different levels of detail for different structures in the city. This means that we can not make any exact comparisons with regard to e.g. how large or how fast cities can be created on different systems. We can however draw conclusions from the type of algorithms used or in what way the technique operates. On the other hand this is made harder by the fact that we do not have access to systems using other techniques. This forces us to choose between either just specifying the data for *Urban* or making our comparison with regards to what is possible to deduce from articles presented by the different system creators.

Other techniques previously explored in this domain are based on a more complex input. Most frequently it is the need for a population density data map used by the techniques to adjust the density of roads or the size and type of the buildings. In comparison, *Urban* has no need for such exact data, which seriously hampers the randomness of the generated result. Instead the residential agent is given a distance value indicating where the latest arrivals to the city would like to live, in relation to the city center. By changing a probability density function and mapping it to a distance value it is possible to emulate a city population with different wants and needs, which has an impact on the density and layout of the city.

Another consequence of the fundamental difference between the simulation-based approach to city modeling of a MAS, compared to other techniques,
which take an approach of *once-over modeling*, is that a MAS based technique is the only one which allows one to see the city during all stages of its lifetime. In figure A.3 a series of images depict the 2D representation of a city during it’s evolutionary process.

Compared to the MAS described by Lechner et al. in [4] where environment data is stored in a grid of land patches, our implementation is based on a data-structure environment collecting different types of urban objects into a corresponding list. This allows the agents to base their decision on an overview of the available choices, enabling them to make the optimal decision in relation to our implementation. The cost of such “more intelligent” decisions comes in two forms. Firstly the environment data structures are more complex and sorting these and finding the best choice is costly. Secondly this method of centralizing the environment data makes the system less able to utilize a multi-threaded solution.

On the other hand if the goal is to depict the real world actors involved in developing a city, it is quite likely that they base their decisions on an overview of the city and not just a local perspective. There is of course times where it is the objective of an agent to act and develop in a specific part of the city but as of yet *Urban* has no way of emulating such a particular need.

### 4.1 Extendability and adaptability

Because each agent is autonomous and can be written as a separate body of code it is very easy to manipulate an agent should it be clear that its behaviour does not emulate the corresponding real-world actor. Because *Urban* only utilizes agents manipulating the environment and not agents manipulating other agents it is easy to detect any erroneous state of the environment. However if an anomaly is the aggregated result of multiple actions, sometimes originating in different cycles by different agents, it can be hard to track the origin. In order to find the origin of such anomalies and to verify that the system is behaving reasonably a MAS tool needs a comprehensive logging and tracking system.

The most apparent drawback of this heavy reliance on interaction between agent and environment, is that there is often a feedback effect into the code of the other agents, is that if there is a need to change the way in which the environment is represented, i.e. because an agent needs more data upon which to base its decisions i.e. because of changes to one agent we are forced to adapt or update other agents. During the development of *Urban* the need for such changes was quite extensive because of the exploratory nature of the chosen software engineering methodology.

Another consequence of not knowing exactly which agents were needed to model a MAS which generated plausible results, was that we had problems finding the right place for the code, i.e. is this specific for one agent or
something more general used by multiple agents. As our familiarity with the problem domain grew this became less of a problem.

Based on the observations in the two last paragraphs the conclusion is that with a clearly defined environment with high enough detail and a good code compartment strategy it should be possible to almost completely remove the need to correct other agents because of changes in one agent.

4.2 Performance and stability

An important aspect of utilizing a urban environment generator with feedback effects between urban objects is that it will always consume more computational power than that of once-over methods. This is because of the strategy of removing old buildings that have been made obsolete with regards to the newer buildings built around the old building. As the city has evolved the reasoning and action that once created the building is no longer valid and therefore the area in question should be recycled by the MAS.

The factor of rework, i.e. the ratio between the sum of all buildings created over the entire time of the simulation and the number of buildings existing, is proportional to two factors; the population increase rate and average building lifetime. If the population increase rate is lower and the goal is to create a city of a certain size, more buildings will be recycled during the production of the city. The inverse relationship is true for building lifetime. The higher it is the less buildings will be needed to replace those that are torn down.

In figure 4.1 the time it takes for the residential agent to find a suitable position for a new land area is plotted against the number of inhabitants in the city. The graph shows the evaluation time during a typical simulation where the city size ranges between 0 and 100 000 inhabitants. From the graph it is possible to see that the evaluation time for the agents increases in proportion with the city population. This is because we allow the agents to consider all possible positions adjacent to already existing areas making the time complexity of the agents reasoning phase increase linearly in proportion with the city size and its population.

*Urban* utilizes a very simple conflict-resolution strategy of the round-robin type. This strategy is one of simplest available and avoids conflicts rather than resolves them. As long as there exists only a small number of agents the round-robin strategy is quite sufficient as it allows us to set the order in which the agents are allowed to act. If however the number of agents would increase the differences between them would get smaller and more agents would have a need for the same resources. Under such circumstances the round-robin strategy would become a priority list giving some agents an unfair advantage over others, thereby possibly distorting the simulation's end result.
The round-robin strategy is also a bad choice with regards to performance. Because of the turn-based style of the strategy the possibility of utilizing a multi-threaded solution is hindered. This is unfortunate since a MAS because of its nature is one of the techniques most suited to take advantage from a multi-core system. Even if a strategy was implemented that allowed the usage of a multi-threaded MAS it is however unclear if the gain would be high enough to warrant the change, since memory usage would remain a bottleneck in the system.

Another problem is the increase in complexity which would be needed to implement some of the strategies, e.g. the bidding strategy, described in section 2.1.1, which would require the implementation of an entire monetary system. Keeping such a system realistic or at least proportional to its real world counterpart would require even more research by the creators and maintainers of the system but might be needed if the goal is to attain another level of realism.

A final aspect of the stability of Urban concerns its ability to avoid anomalies. The most evident fact here is that the MAS technique does not supply any mechanism which guarantees the user that the created city is free from anomalies. Their existence is entirely dependent on how well the agents are implemented. As the complexity of the agents and environment increase so does the risk of introducing anomalies.
4.3 Realism

In the case of a generated urban environment, the perception of realism is a result of the expectations of the user compared to the actual artificial environment. Based on this two conclusions can be drawn:

1. Different people will perceive an environment more or less realistic depending on their understanding of the dependencies between urban objects and their understanding of the factors that govern how the city evolves.

2. When trying to mimic the developmental behavior in a MAS the success rate is dependent on understanding the actors, the ability to extract the characteristics of their behavior and implementing it in the artificial agents.

This means that there are a lot of enhancements which could be implemented should more realism be of importance. The key is to select such enhancements which will effectively add realism and plausibility to the city but does not require a large increase in system complexity.

Another important fact to note is that the cities we generate might be realistic to one group of people but not to another and that the best way to guarantee that the result is perceived as realistic is to create an adaptable environment generator.

The only way to actually be able to tell whether Urban somehow has provided realistic results in itself or comparing to other methods is to expose it to subjective testing. This could for instance be a testing session in which randomly selected individuals are subjected to environments generated by Urban and by other tools and then comparing how the reactions differ between the different environments. From this it would be possible to deduce which method creates the most realistic environments. Unfortunately such a test session is not possible to perform in the short period of this thesis.
Chapter 5

Conclusions

5.1 Summary

The problem of generating realistic cities has emerged as a natural step in simulating the world. Whether it has been in order to perform simulations to test or for entertainment reasons such as gaming. Even though the task is still relatively new various different approaches to solve it has emerged.

Most solutions have used a procedural approach, these have emphasized on generating the city by adding suitable objects to suitable places. This has been based on road structure or density maps. The approach towards AI these methods are using is sometimes called acting rationally. They generate something that looks like it has been created with rational thought.

In contrast to the techniques mentioned above we have MAS that has a thinking rationally approach to AI. An agent is an actor that at each step of decision making will look at the current state of the world. It will then take its sets of rules and procedurally run through them to apply them on the world.

The multi-agent system is a system that consists of several independent agents. All these agents will work on the world until each one is satisfied. These changes will then give incentive to other agents to also want to add certain things to the world. What initially starts this reaction is the initial state of the world. The world can be generated in many ways and this initial start influences the whole output of the simulation.

It can also come to the point where all agents are satisfied with the state of the world, unless there is a outbound actor that changes the world.

These are the conditions of a multi agent system. What makes this method more suited for the problem is its adaptability to change. In a world of constant change it is necessary for actors of a system to act on the current state of the world instead of stick to a pre-designed plan.

So MAS is a very suitable approach for generating a dynamic urban simulation because it is possible to emulate the rational reasoning of the
real-world actors in every decision made in the simulation. Urban is the resulting program produced out of these ideas.

Urban is a tool that will visually simulate the development in real time. It had to be developed in such a manner that it would continuously give the user feedback on what was going on in the world. Also since the MAS supports dynamic development it can be interesting to add changeable parameters that will directly influence the reasoning of the agents and thereby the continued output of the simulation.

The major difficulties of developing such a tool is the exponentially increasing amount of data that has to be handled. The world that represents the data needs to be filled with all necessary data in order for the agents to make correct decisions. Also it needs to be possible to access and change the data when e.g. roads can be built from one place to another without any problems.

Besides keeping track of building, land areas, roads, buildspaces and so on, one of the fundamental data representation is that of the pre-urban landscape. The terrain can fundamentally change the output of the whole simulation. So a customizable terrain generator has been required and developed.

There are many levels in which the techniques of Urban can be compared with other techniques. The main points on which the results of Urban differs from other results is its nondeterminism and that it is hardly dependent on input data.

The low dependency of input data enables it to be more automatic. It will lack the realism that these input data bring in the way that they are actually taken from real urban landscapes. The agent will however provide much more unique cites in sense that they do not have any basis to run on except for adapt to explicit data. The programmed rationality may be adequate to fool people in sense of realism. This is of course dependent on how well they are characterized.

The use of MAS also leads to other benefits, such as its flexibility when it comes to extending and adapting the system or parts thereof. This mainly concerns agents which can be altered totally independent of each other.

The real problem of MAS comes when one realizes that the environment made for the agents is inadequate for further development. If perhaps a agent crucially needs some additional information then the world must be rebuilt. And if one is really unlucky one has to make adjustments in all agents to handle the additional information.

5.2 Future work

During the development of Urban many ideas were hatched. Very few of these actually got implemented and even fewer are in use today. Realistically
all ideas could not be implemented within the short range of the thesis. Therefore there will be a list of some of the many ideas that were thought of but could not take place in the resulting application.

One of the most obvious improvements would be to simply improve the agents. This mission will have to be based on observing the real world and determining cause and effect in urban architecture.

But there are problems that occur when using this approach. If we add our improvements solely to find the correct piece of land the problem will be so complex it will be unmanageable. But if the improvement is cleverly integrated into the agents as well as the LandProjector great improvements can be made.

- Roads and streets should not share data structures. While there exists both a street agent and a road agent they produce roads of the same size and classification. To simplify agent reasoning the structures should be separated allowing more complex wants to be modeled. This could for instance be residential areas which need road connections but usually should not be near highways where there is a lot of noise and pollution. The latter is not true for industrial areas which should be placed near to primary roads in order to receive and ship supplies.

- Traffic on the roads could be calculated to give the road agent more information upon which to base its decisions. For instance if a road downtown is too heavily used something will usually be done to address the problem. The same should happen in a virtual world. There are plenty of possibilities such as broadening the road, creating beltways around the city, adding more public transportation or even changing the location of a heavily visited public facility. All these options would add to a more dynamic world and the options could even be controlled by a input parameter.

- Some work has been made on creating suburbs in the end of this project but enough time was not available to perfect the feature. In the current state residents are widely spread out. But the rest of the community do not follow this trend as well as they should. This would help to decrease the traffic of the town and also add housing with quieter surroundings which appeals to some people.

- Originally there were a lot of thought about making the land planning more dependent on the landscape. Meaning the land projector would understand that some parts of the map are more or less suitable to build on. For instance building on sloping land would be much more expensive than to build on flat land. Or that coastal lines are more attractive for residential areas. This mechanism could act as both incentive and deterrent for the agents. An example of the latter could
for instance be that it would be costly to chop down existing forest in order to build something in that spot. Reversely, the same forest could act as an incentive for an industrial company to build a lumber shipping facility near the forest.

- The type of road strategy used is a very important factor which determines what kind of look the city will have. There is everything from strictly squared Manhattan style to purely organic road structures or anything in the range between these two. A city can have a strict square pattern center with organic looking road structures stretching out from it. It could also be the other way around, i.e. where an organic strategy is used for the center because early inhabitants of the city were forced to adapt to the terrain instead of like today adapting the terrain to their needs.

The road strategy could be parametrized in one or more variables allowing the user to control the layout of the road network in different parts of the city.
Bibliography


Appendix A

Image results

Below is a compilation of different cities in 2D and 3D generated by *Urban*. The results will best be shown in form of images.

Figure A.1: Overview of a mid-size nucleated city in 3D viewing mode.
Figure A.2: A city shown at different points of development. a) Year: 139, Pop: 5121
b) Year: 163, Pop: 10393 c) Year: 186, Pop: 20495 d) Year: 209, Pop: 40432
Figure A.3: Same city as in d) above but in 3D.

Figure A.4: A city with a tight distribution and no neighbors.
Figure A.5: A city where the trees are rendered.