UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Evaluation of Injected Data from an Adapted Smart City Simulation

Evaluation von importierten Daten anhand einer adaptierten Smart-City-Simulation

Bachelorarbeit

verfasst am
Institut für IT-Sicherheit

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Kilian Zeiseweis

ausgegeben und betreut von
Prof. Dr. Esfandiar Mohammadi

Lübeck, den 23. September 2021

IM FOCUS DAS LEBEN

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Kilian Zeiseweis

Zusammenfassung

In einer Gesellschaft, in der Digitalisierung und Vernetzung in unserem Alltag immer präsenter werden, gewinnen auch „Smarte Städte“ zunehmend an Bedeutung. Ein großes Problem hierbei ist jedoch die Privatsphäre der BürgerInnen, vor allem bei dem Beziehen von Live-Realdaten. Um dieses Problem anzugehen, wird in dieser Arbeit das Computerspiel Cities:Skylines verwendet, um eine Smart-City-Simulation zu erstellen, welche einen Fokus auf das Schützen eben dieser Privatsphäre legt.

Im Rahmen dieser Arbeit wurden insgesamt zwei Spielmodifikationen, sogenannte „Mods“, entwickelt. Während die erste Mod detaillierte Verbrauchswerte wie Strom, Wasser, Heizungswärme, Abwasser und Müll sowie Verkehrsdichte und Lärmemissionen innerhalb des Spiels extrahiert, passt die zweite Modifikation die Stadt in eben einem dieser Parameter an externe (Real-) Daten an.

Zudem wurden im Rahmen dieser Arbeit drei Experimente durchgeführt. Das erste überprüfte oberflächlich den Realismus der unmodifizierten Simulation mit Hilfe von Sanity-Checks. Dieses ergab, dass mehrere der oben genannten Verbrauchswerte die meisten dieser Checks bestanden. Es zeigte jedoch auch, dass die Verkehrsdichte und die Verkehrslärmemissionen in dem Spiel noch weit davon entfernt sind, realistisch zu sein. Das zweite Experiment zeigte, dass der Stromverbrauch des Spiels mit Zufallswerten angepasst werden kann. Diese Anpassung funktionierte, jedoch traten hier vermehrt extreme statistische Ausreißer auf. Im dritten und letzten Experiment wurde dann die Stadt an den aggregierten Stromverbrauch der realen Welt angepasst. Dieser wurde dankenswerterweise von der TraveNetz GmbH in Form von aggregierten, realen Verbrauchsdaten zur Verfügung gestellt. Im Gegensatz zum zweiten Experiment konnte die Simulation signifikant besser angepasst werden, ohne dass solch extreme Ausreißer auftraten. Insgesamt kam es hier zu einem durchschnittlichen Fehlerwert von etwa 11%. Das letzte Experiment zeigte vielmehr, dass Cities:Skylines als Simulationsumgebung mit einer sehr geringen Datenbasis anpassbar ist, ohne die grundlegenden Eigenschaften der Simulation verletzt wurden. Alles in allem kann man daher sagen, dass das Spiel einen vielversprechenden Ansatz liefert, um eine Smart-City-Simulation mit Hinblick auf die individuelle Privatsphäre zu erstellen.

Abstract

In a world where digitalization and digital networks are becoming more and more present in our daily lives, smart cities are also becoming increasingly important. This work uses the computer game Cities:Skylines to create a smart city simulation with respect to individual privacy. During this work, two modifications were developed for the game. While the first one extracts detailed consumption rates, such as electricity, water, heating, sewage, and garbage along with traffic density and traffic noise emission, within the game, the second one adapts the city in one of these parameters to external data. Furthermore, three experiments were conducted during this thesis. The first one checked the overall realism of an unmodified Cities:Skylines using common-sense sanity checks. This experiment revealed, as a result, that several consumption rates listed above are passing most of these sanity checks. However, it also reveals that the traffic density and traffic noise emissions are far from being somewhat realistic in this game. The second experiment showed that the game's electricity consumption could be adapted using random values. This adaption worked out okay, despite some heavy statistical outliers. The third and last experiment adapted the city to aggregated real-world electricity consumption thankfully provided by the TraveNetz GmbH. In contrast to the second experiment, the simulation was able to be adjusted even better without severe outliers, resulting in an average error of about 11%. From the privacy point of view, this last experiment is of particular interest. It shows that the simulation can be injected using a small real-world data basis but keep its simulated properties. All in all, this work shows that Cities:Skylines is a promising candidate to create a smart city simulation without violating individual privacy.

Acknowledgements

At this point, I would like to thank Prof. Dr. Moreen Heine from "Institut für Multimediale and Interaktive Systeme" and André Gode from MACH AG for their support during this work. Furthermore, I would like to thank Arne Heinrich and the whole TraveNetz GmbH for their support and the provision of real-world electricity and water consumption data to compare and adjust the simulation.

Contents

1

Introduction

Nowadays, it is hard to find any part of our society that is not affected by digitization or digital networking. Moreover, as new developments of intelligent devices are constantly finding ways into our daily lives, smart cities are becoming more and more present too. Smart cities, in general, are a broad field that can affect almost every part of the city administration, e.g., smart governance, smart transportation, and smart IoT sensors. Fanghao Yu et al. showed in their case study how the LoRaWAN protocol could be used for smart city applications, for example, to gather real-time air quality information [20].

Planning a smart city is very challenging, nevertheless. Especially modern cities are getting more and more complex. In particular, so-called "what if?" questions [14] are of interest, as these could show how a single change impacts the whole city. If the planning authority, for instance, wants to build charging stations for electric cars, they need to know how this will impact the electricity consumption of a particular region.

The most forwarding approach would probably be to create a machine-learning algorithm and train it with real-world data. Afterward, the resulting artificial intelligence can predict the impact due to changes. This approach has two disadvantages, unfortunately. First, an extremely high amount of data is needed to train the AI properly. This data needs to be collected due to a high amount of sensors all over the city. Moreover, private data can be reconstructed from trained machine learning models [2], which is the second disadvantage. Both cases can lead to significant privacy violations. A malicious person could manipulate a sensor to steal personal data, as Abosaq et al. pointed out [1]. This danger could be countered by protecting the data using modern cryptography. Nevertheless, there is a risk that these data can be read.

This privacy problem can be bypassed by using smart city simulations. In general, a simulation can be seen as some kind of black-box modeling a simplified version of the desired city internally. Thus, the parameters used to simulate the city can be adjusted easily. These parameters can be anything based on the simulation environment, so a simulation could help answer the "what if?" questions mentioned above. Moreover, a simulation already implements how a city works. Therefore, it does not need as much data as machine-learning techniques.

However, using a simulation has some challenges, too. The main disadvantage is its inaccuracy due to less real-world data. Since the logic is already implemented, it is not as precise as a machine-learning model trained with data just of the desired city.

Finally, the primary purpose of this thesis is to investigate the computer game Cities:Skylines as a simulation environment. Furthermore, this work will show how detailed data can be obtained within the simulation and how real-world data can fine-tune it. There is still a privacy risk due to these external values. Therefore, this approach works with statistical data that is far easier to protect, for example, by using differential privacy methods [7].

1.1 Contributions of this Thesis

The primary purpose of this thesis was to find a way to export and import data from respectively into the simulation environment Cities:Skylines that was improved with several game modifications (so-called “mods”). Therefore, two modifications with the related tasks were constructed, developed, and deployed. These mods made extracting and importing consumption and emission rates, such as electricity, water, sewage, heating, garbage, and traffic noise density. However, it did not change either the traffic density or the citizen density, although both of these parameters can be exported too.

Moreover, three experiments were conducted: The first was designed to evaluate the overall simulation’s performance without importing any data. This performance test was realized by exporting data from within the simulation and counterchecking them with so-called “sanity checks”. Doing so revealed a mixed image, as the simulation fulfilled some of the sanity checks and some did not.

The second experiment tested the performance of the simulation fed with randomly generated electricity consumption values. After importing these random values and adapting the simulation to them, one week of electricity consumption was exported and compared with the imported data. Despite some heavy outliers, the exported data fitted to the imported ones.

The third experiment then imported average real-world electricity consumption rates into the game. Afterward, another week of electricity consumption was exported, revealing quite promising results. In this case, the exported data fitted better to the imported real-world dates without showing heavy outliers and a Mean-Average-Percentage-Error (MAPE) of about 11%. Moreover, it shows that Cities:Skylines can be adapted using a minimal data basis of seven values without losing elementary simulation properties.

1.2 Related Work

Duncan et al. showed in their bachelor thesis how Cities:Skylines could be used for urban planning optimization [6]. Therefore, they implemented a bridge to a neuronal network into the game. The neuronal network then builds a city using Cities:Skylines, based on a parameter map. As a result, they trained a remarkable artificial intelligence that is capable of realizing an urban plan based on the parameters. However, in their work, they did not import any real-world data.

Olszewski et al. created a serious Cities:Skylines save game based on official topological data [14]. In more detail, the authors implemented this Cities:Skylines save game from solving social and ecological problems. Therefore, they created a modification to import a Polish region of about 135 km^3 containing many poultry and pig farms in that area. As this region lacks many environmental problems due to the farms, they also simulated the impact of biogas plants as a solution. However, their work gives a detailed basis for further use of Cities:Skylines in the same way this work does, but their work does not focus on the data protection discussed above.

Suciu et al. explain a smart city simulation solution apart from Cities:Skylines [19]. In their work, they focus on a 3D city model that gets fed with crowdsourced traffic data. This simulation can be used for urban planners to investigate changes or interventions in road traffic. In more detail, the traffic data is the position, time, speed, and orientation of smartphones from participating users. Therefore, their work highly depends on a massive amount of publicly available data.

Last but not least, Karnouskos et al. designed a smart city simulation based on publicly available average electricity consumption [11]. Therefore, they obtain their data from average appliances' consumption that can be found in regular households. Moreover, their simulation can also simulate electric cars as a significant part of a city's electricity consumption. Apart from average electricity data obtained from studies, no real-world data comes into the simulation.

1.3 Structure of this Thesis

After this introduction, the thesis is divided into two main parts. The first part includes chapters 2 and 3 explaining the general modding process and the development of the two modifications created for this thesis. The second part (chapters 4 to 6) contains three experiments. The purpose of all three experiments is to validate the simulation and its adaption using the previously developed mods. The first experiment checks the overall realism of Cities:Skylines. In the second experiment, a set of random data gets imported as the electricity consumption. Finally, the electricity consumption is exported and compared with random values imported after the import is finished. In the third experiment, the electricity consumption for an average day gets imported into the simulation. Afterward, the electricity gets exported and compared to the real-world data again. Finally, the last chapter, 7, summarizes and discusses the results.

2

Cities:Skylines as a Simulation

2.1 Preliminaries

The program of choice was Colossal Order's computer game Cities:Skylines, published by Paradox Interactive in 2015 [4]. The game operates as a simulation environment for the city of Lübeck. Therefore, the simulation comes as a save game of Cities:Skylines. Loading the save game from within the game will start the simulation process.

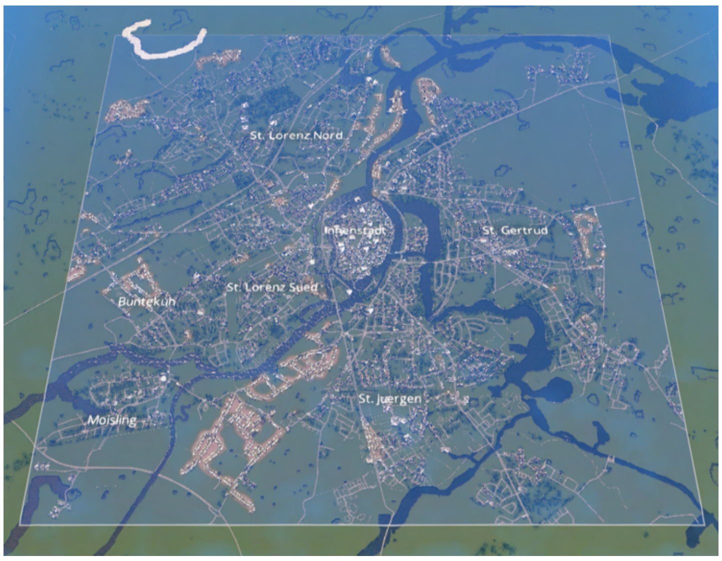
The save game used here was initially created by a modification called "GeoSkylines" within the work of Richter [18]. In general, this mod takes data from OpenStreetMap and creates a save game from it [15].

To get the simulation run properly, many mods are needed. These mods are necessary to break the game logic at some points. A list of these mods can also be seen in Richter's work [18]. As the Cities:Skylines is primarily designed to be a computer game, its original purpose is to entertain its users. Therefore, the game is getting more and more challenging while the simulation progresses. It implements several mechanisms to catch the user's attention. For example, different hazards such as fires or floodings can occur on which the player has to react. Furthermore, such behavior would completely break the simulation flow and is disabled by mods, therefore.

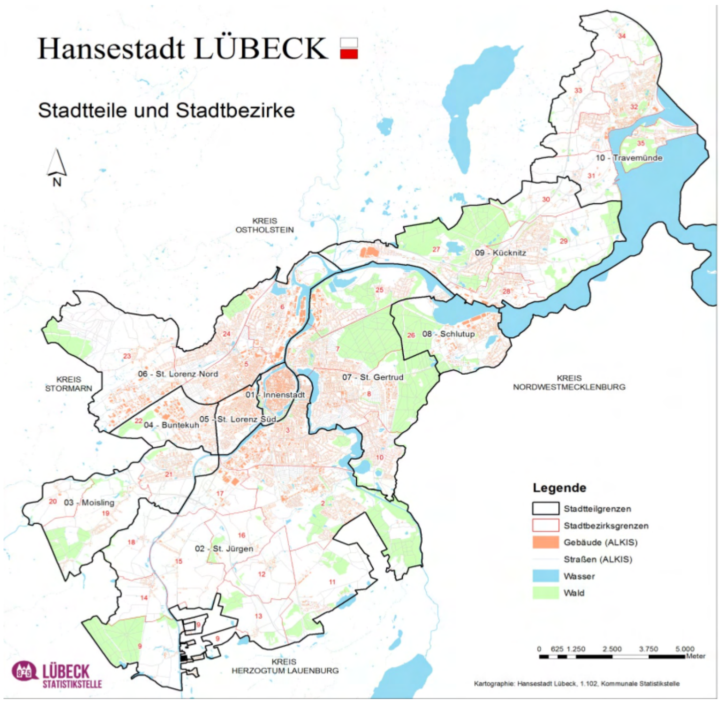
2.2 General Setup

However, not all parts of the save game used in this work were created by GeoSkylines. For example, the districts shown in figure 2.1a were added per hand using the game's own district manager. Thereby, the second map shown in figure 2.1b was used as a template. The district manager is essential as the two mods developed for this work are highly taking advantage of it. Furthermore, the game offers the opportunity to add the districts manually using a paintbrush-like tool. This tool is extremely powerful as the districts added manually are not restricted in their size. Finer graded divisions of the city are also imaginable, for example, a division by postal codes.

One more thing added to this thesis is the RealTimeMod, which will be discussed in the next section.



(a) Overview of the Lübeck city simulation by day. A noticeable detail in this picture is the white writings lying over the city. These indicate the administrative districts of Lübeck. However, these districts were added by hand using the paintbrush-like in-game district manager and are not imported from OpenStreetMap.



(b) Political map of Lübeck city [3]. This map was used as a template for marking the districts in the simulated map above.

Figure 2.1: The two figures shown here are maps of the Lübeck city area. While the upper one shows the view from within Cities:Skylines, the lower map shows the real-world administrative districts of Lübeck. Thus, in a superficial comparison from above, both maps look somewhat similar.

2.3 RealTimeMod

The RealTimeMod is a Cities:Skylines modification added in the context of this work. Its primary purpose is to make the game a little bit more realistic.

One problem of the original game is the lack of a timetable for the citizens. This means, for example, that the citizens though are going to work or school, but they do not have regular times for it. The game developers probably implemented this mechanism in order to require less CPU power while playing the game. However, the RealTimeMod solves this problem by simulating each citizen with a somewhat schedule [16]. This changes the behavior of the simulation a lot. For example, this adds timing-based events like rush hours to the traffic flow, as citizens now start working together at almost the same time. Furthermore, this mod adds weekends to the citizen's schedules. At weekends, citizens neither go to work nor school.

Figure 2.2 gives an instance of how deep the changes of the RealTimeMod go. Both graphs show the hourly electricity consumption of the district St. Lorenz Nord on a random Saturday.

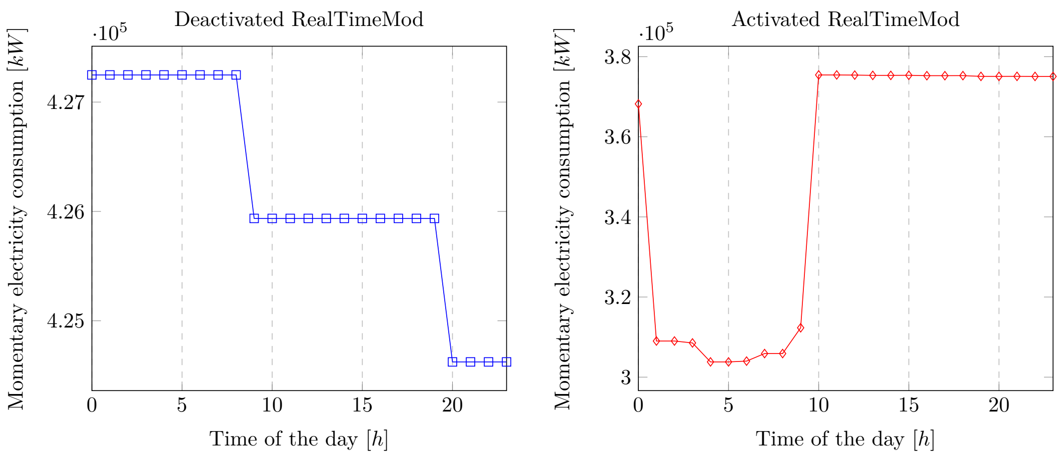


Figure 2.2: These two diagrams show the electricity consumption of the district St. Lorenz Nord for a random Saturday. Therefore, the left diagram shows the consumption during a simulation with no activated RealTimeMod. The chart on the ride side was recorded with the RealTimeMod enabled. However, comparing it reveals a significant difference in the handling of the electricity consumption. The complete evaluation of the RealTimeMod will be part of the experiment in chapter 4.

However, there are two primary disadvantages when using the RealTimeMod. The first one is the significantly higher power consumption. To simulate a city with the size of Lübeck (with approx. 227,000 citizens), a lot more computing performance is required. The second disadvantage is the simulation's speed. In the original version of this mod, the speed only can be set between a value of one and six. While preset one is almost the real-time speed (meaning a real-time minute equals nearly a minute of the simulation), even preset six is way slower than the original Cities:Skylines. An hour in the simulation then lasts $168.172ms$ in real-time, which is ≈ 2.8 minutes. Recording ten weeks of data from the simulation would last $7.84h$.

Since the code of the RealTimeMod was published under the MIT license, it can be downloaded, modified, and recompiled locally. Therefore, the presets can be adjusted in code to enable further speed settings. However, this should be used carefully. Speeding the simulation up requires very fast hardware and can break the simulation if the value is set too high. This work set the speed to level eight in each experiment, which simulates one hour by $\approx 36s$. In an unmodified Cities:Skylines, an hour gets simulated in $\approx 700ms$.

The next chapter will show how mods are built in general and how they can export or import data.

3

Modifying Cities:Skylines

One main advantage of using Cities:Skylines as a simulation is the adaptability using a modification. These mods allow the development of code that will be executed during runtime. Moreover, these mods can read data from inside the simulation or change the game's behavior.

3.1 Modding-API

A modification in Cities:Skylines is mainly a compiled C# library that hooks into the game code. This can be done by calling in-game functions or inheriting in-game classes. In general, there are two ways of letting the game execute code from outside. The first method is using the official modding API Colossal Order provides [5]. The second one is to use C# reflections to overwrite game functions [17]. However, the compiled game code itself is neither human-readable nor documented, apart from the exported function names. Thus, developing software offside from the official modding API is not trivial at all.

As indicated above, a game modification must be written in the programming language C#. The game itself was written in C# too. Furthermore, the game seems to take advantage of the manager design pattern [12]. Many parts of the game, such as districts, traffic, power delivery, roads, buildings, and many others, are organized within a manager class. These managers and their subclasses are themselves arranged in a pattern shown in figure 3.1. Understanding this pattern makes it therefore easy to create mods for a large part of the game.

The following example will explain figure 3.1 using the `Building` class of Cities:Skylines. The `BuildingManager` organizes every building within the game. As shown in the figure, it inherits its main manager functionality from the `SimulationManagerBase`. Additionally, the `BuildingManager` has implemented the `SimulationStep` methods, which are two in this case. These methods seem to get called as soon as the simulation is doing its job. Furthermore, the BuildingManager has a `m_buildings` property, which is a `Building` typed `Array16` object. Internally, this array object has a property called `m_buffer`, which is the actual array containing all simulation building objects. Moreover, the `Building` class has a `BuildingInfo` object containing additional logic for each building. Finally, one of the further logic of the `Building` class stored in the info object is the `BuildingAI`. This artificial

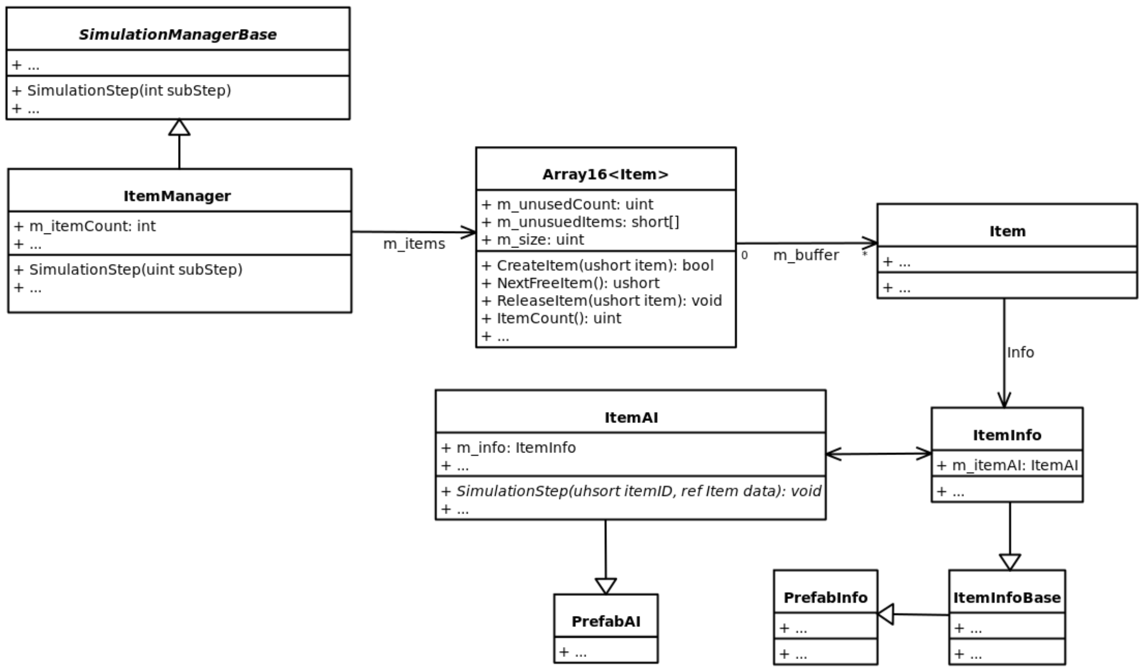


Figure 3.1: This simplified UML class diagram gives an overview of the manager design pattern within Cities:Skylines. Each class is provided by game libraries. Every “Item” shown here stands for an object of the game used internally, for example, districts, traffic, power delivery, roads, buildings, and many others. However, this overview is very simplified: Each class in this graph has a lot more properties, methods, and super- or subclasses (indicated by the “. . .”) which are left out here due to clarity. An example regarding this infrastructure can be found in section 3.1.

intelligence type belongs to the `PrefabAI` and implements its own `SimulationStep` method.

The following section will explain the usage of this architecture to extract data from within the simulation.

3.2 Extraction

The last section outlined how modding works out in Cities:Skylines. This section will now take a closer look at how to extract detailed data from within the simulation. Furthermore, this will lay the foundation of the CSExtractDataMod in section 3.2. Before start extracting things from the simulation, the objects of extraction must be defined. As the consumption and emission of citizens is the main focus of this work, the following parameters were chosen:

1. Electricity consumption
2. Water/Sewage consumption
3. Heating consumption
4. Garbage emission
5. Traffic density
6. Traffic noise emission

Electricity, water, sewage, and heating are of particular interest for smart city simulation, as supplying it needs a very complex infrastructure. Modeling the consumption and emission of these parameters can help to find inefficiencies in the supply chain. Therefore, testing and evaluating these parameters will be the subject of the investigation in chapter 4 and the following. However, there are two options for extracting the first four key points out of the game. The first one is to use the `BuildingManager` explained before in section 3.1. Here, the idea is to iterate over each building (via the `m_buffer` array shown in figure 3.1) at a specific time and sum up the consumption respectively the emission rates. Unfortunately, this option requires using C# reflections to hook into the calculating method manually, as there is no official API call returning the requested values. The second option is to use the `DistrictManager`. Fortunately, districts in Cities:Skylines are organized just like in figure 3.1. Furthermore, the `District` class contains official API methods to receive each parameter aggregated for each district. Listing 3.1 shows the second option implemented in a C# method.

Listing 3.1: C# method to read and print data for each district of the city. The `Debug.Log` environment allows printing a string within the debug window of Cities:Skylines.

```csharp
// Prints the electricity, water, and heating consumption
// along with the sewage and garbage emissions to the
// debug console.
public void PrintDistrictData() {

  // Get the district manager by its static instance.
  DistrictManager districtManager = DistrictManager.instance;

  // Get the actual district array.
  District[] districts = districtManager.m_districts.m_buffer;

  // Iterate over the districts. Important: the district
  // at position 0 in the array is always the city itself.
  for (int i = 0; i < districts.Length; i++) {

      // Get the district object from the m_buffer array.
      District district = districts[i];

      // Print the name of this district.
      Debug.Log(districtManager.GetDistrictName(i));

      // Print the requested values of this district.
      Debug.Log(district.GetElectricityConsumption());
      Debug.Log(district.GetWaterConsumption());
      Debug.Log(district.GetSewageAccumulation());
      Debug.Log(district.GetHeatingConsumption());
      Debug.Log(district.GetGarbageAccumulation());
  }
}
```

Regarding traffic and noise density, the situation is not much different. Both are essential when it comes to traffic flow optimization. Furthermore, a previous work pointed out that the noise emission is somewhat comparable with the fine dust air pollution [18]. Extracting traffic-related data works very similarly to exporting data from a district. In fact, the roads are handled by the so-called `NetManager`, which works the same way as the `BuildingManager` and the `DistrictManager`. This `NetManager` contains an array of segments. Each segment represents a part of a street and connects two crossings. Now, the same algorithm used in listing 3.1 can be adapted to iterate over the array of segments. The segment class parameters `m_trafficDensity` and `m_noiseDensity` will for each array element contain both traffic and noise density.

Testing and evaluating all of the data will be the subject of the investigation in chapter 4 and the following. The next section will show how these techniques can be used to export data from the simulation into a CSV file.

CSExtractDataMod

The CSExtractDataMod is a Cities:Skylines modification which is meant to be a recording tool. It was developed due to this work to add the possibility of extracting data from within the simulation. Therefore, it is based on the algorithm from listing 3.1. The user can set a tick at a topic in the menu he or she wants to export.

The mod then starts the recording process. A recording here is primarily a hash map that consists of two types: string and integer list. Hence, the string contains the identifier of the current recording property (e.g., when recording districts, the string holds the district name). The integer lists, on the other hand, consist of the actual values that will be exported. The sense behind using a hash map is the access time. Searching for an existing string in a hash map and adding values to a list happens in $\mathcal{O}(1)$, which is very useful when handling datasets with unpredictable growth. Table 3.2 gives an instance of what those hash maps look like.

Key	Value
Lübeck	$[1590064, 1327328, 1327328, \ldots]$
St. Gertrud	$[359937.6, 360009.6, 359888, \ldots]$
St. Jürgen	$[463052.8, 364408, 357011.2, \ldots]$
⋮	⋮

Table 3.2: Example for a hash map generated by the CSExtractDataMod. In this instance, the user recorded the electricity consumption for more than three hours, as shown in the lists in the “Value” column. Therefore, the “Keys” of the hash map are the names of the districts. While the recording continues, the new consumption values are added at the end of the list for each district.

As shown in the section before, it is relatively easy to read the desired data from the simulation. As a result, the CSExportDataMod needs a trigger which calls the method from listing 3.1 and automatically adds the data to the ongoing recording. The official modding API, more precisely the `ThreadingExtensionBase`, can be used to achieve this. However, according to the modding API documentation, the methods in the `ThreadingExtensionBase` are getting called 60 times per second when playing at normal game speed. Therefore, the following code listing 3.2 shows the method `OnBeforeSimulationTick`, which uses the modding API to override the virtual method from its superclass and thus hooking into the simulation process with 60 calls per second.

Listing 3.2: C# class to add data to all ongoing records whenever a full hour was reached.

```
// Modified ThreadingExtension is hooking into the simulation threading
// process using the official modding API.
public class ThreadingExtension : ThreadingExtensionBase {

  // A private threading class is used to get the current simulation time.
  private IThreading Threading = null;

  // The next hour, on which will be a recording.
  // Initializing it with 0 indicates that the next
  // recording will start on the next day.
  private int NextHour = 0;

  // The OnCreated method gets called when the class
  // gets initialized the first time. In this case,
  // it initializes the threading property before it
  // calls its related base method.
  public override void OnCreated(IThreading threading) {
    this.Threading = threading;
    base.OnCreated(threading);
  }

  // This method is the reason for this class. It gets invoked
  // 60 times per second and handles the ongoing recordings
  // before it calls the base method of the superclass.
  public override void OnBeforeSimulationTick() {

    // The recordings are getting updated, whenever
    // there is an ongoing recording, the simulation
    // time reaches a full hour, and the simulation
    // is not paused yet.
    if (CSExtractDataMod.IsRecording() && Threading.simulationTime.Hour == NextHour
        && !(Threading.simulationPaused)) {

      // For each ongoing record, the data at the current
      // simulation time gets added.
      foreach (Recording record in CSExtractDataMod.GetAllRecordings()) {
        record.AddRecord(Threading.simulationTime);
      }

      // The nextHour parameter is set to the
      // next hour on the clock.
      NextHour = ((NextHour + 1) % 24);
    }

    // Finally, the base method gets called.
    base.OnBeforeSimulationTick();
  }
}
```

In theory, much higher recording rates than hourly are possible. But, the problem which must be kept in mind when using the `ThreadingExtensionBase` is that every code getting executed within the `OnBeforeSimulationTick` method effectively delay the simulation process. Thus, if the code inside the condition of the method is executed more often, the simulation can be delayed resulting in unpredictable side effects. Furthermore, if the time interval between two recordings is too short, the data in the target class may not have been updated in the meantime. In the end, this can lead to inconsistent data.

Afterward, the data from the hash map gets converted into a comma-separated-values (CSV) string.

3.3 Injection

The last sections showed that extracting existing data from the simulation is not as difficult because of the modding API. Now, things are getting more complicated when it comes to injecting data into an existing simulation. Several circumstances make it particularly difficult. At first, injecting existing data into simulation is not covered by the official modding API. Therefore, C# reflections must be used to inject code into the game libraries. Second, the code is closed source. It is unreadable, and there is no documentation about it. Because of this, developing under these circumstances follows the principle of trial-and-error, making it very hard to code productively. However, there is a way to change the consumption and emission properties listed in 3.2. The `Building` class shown earlier has, following figure 3.1, a `BuildingInfo` object. This `BuildingInfo` object again has the property `BuildingAI`. Each "ItemAI" class itself has a series of subclasses, so does `BuildingAI`. Each subclass defines a finer graded logic for their related "Item" class. E. g., the `ResidentialBuildingAI` inherits from the `BuildingAI` class and implements a finer graded artificial intelligence just for residential buildings. However, one class in the subclass hierarchy of the `BuildingAI` is the `CommonBuildingAI`, handling almost every type of building inside the simulation. Moreover, this class implements a method called `HandleCommonConsumption`. As the name already suggests, this method handles the consumption of a single building. Furthermore, this method gets the reference to each consumption or emission value listed in section 3.2 passed as an argument.

At this point, the modified code must be injected into the `HandleCommonConsumption` method. This injection can be done effectively using an injection library named "Harmony" [9]. Using Harmony provides a collection of options to inject code into the desired method. In this case, the postfix option was used to inject code directly after the base method was invoked. This option allows preserving the original logic developed by the game developer but changing the results of the method at the same time. A simple proof-of-concept example to this can be seen in code listing 3.3.

Listing 3.3: This simple proof-of-concept shows how data effectively can be changed using C# reflections. This example, in particular, will set all consumption and emission rates to zero, resulting in a city not consuming or emitting anything.

```csharp
// This postfix method gets called directly after
// the original HandleCommonConsumption method of the
// CommonBuildingAI class. Therefore, this method has the
// same parameters passed through it as the original method.
public static void HandleConsumptionPostfix(
  ushort buildingID,
  ref Building data,
  ref Building.Frame frameData,
  ref int electricityConsumption,
  ref int heatingConsumption,
  ref int waterConsumption,
  ref int sewageAccumulation,
  ref int garbageAccumulation,
  ref int mailAccumulation,
  int maxMail,
  DistrictPolicies.Services policies
) {

  // Setting all parameters to null results in
  // a city that neither consumes nor emits anything.
  electricityConsumption = 0;
  heatingConsumption = 0;
  waterConsumption = 0;
  sewageAccumulation = 0;
  garbageAccumulation = 0;
}
```

```

// This method uses Harmony to patch the method above using
// C# reflections.
public void Patch() {

  // Create a new instance of Harmony based on the mod-unique
  // harmonyString.
  private static readonly string HarmonyString =
      "de.uni-luebeck.smart-city-simulation.cs-import-data-mod";

  Harmony myHarmony = new Harmony(HarmonyString);

  // This call patches the method above as a postfix method getting
  // called after the target method from the CommonBuildingAI class.
  myHarmony.Patch(
      typeof(CommonBuildingAI).GetMethod("HandleCommonConsumption",
          BindingFlags.Instance | BindingFlags.NonPublic),
      postfix: new HarmonyMethod(typeof(this).GetMethod("HandleConsumptionPostfix"))
  );
}
```

CSImportDataMod

As the name already suggests, the CSImportDataMod's primary purpose is to take data from outside and import them into the game. It was, along with the CSExtractDataMod from section 3.2, the second modification developed during this work. The small program shown in the last section already proved that it is possible to change the consumption values for each simulation building. Just like in the CSExtractDataMod section, this mod will also focus on electricity, water, heating, sewage, and garbage consumption and emission.

Given the following situation: an external organization provides some aggregated data for each district of the city. The most straightforward approach to inject these data into the simulation would be as follows: Divide the aggregated value of each district by the number of buildings and set the result directly as the new consumption using the algorithm from listing 3.3. This approach would probably work out, but it would also mean a loss of information. Because this algorithm simply overwrites the values the base method calculated before, all dependencies regarding the result will be lost. For example, a bigger house with more citizens would now consume the same amount of electricity or water as a smaller one with fewer residents.

A second approach that avoids this obstacle is the following: Instead of placing the value directly, a factor x between the simulation and the target value from outside will be calculated. This factor then scales the results `HandleCommonConsumption` method up or down. However, it is not recommended to calculate x just once. Doing so can result in a somewhat bad factor, as the results of the `HandleCommonConsumption` method can change over time, based on unforeseen events. Moreover, it is essential to find something like a sweet spot for x so that the results remain stable over time.

The CSImportDataMod is based on this second approach. It will try to find a good x for the external values. The assumption is that the external data contains hourly values of any consumption or emission listed in section 3.2 for one week and each district. In more detail, the data to be imported comes as a matrix with $7 \cdot 168 = 1176$ elements. In fact, the algorithm will not try to find just a single factor x as this would be not target-oriented in most cases. Furthermore, x is here a seven times 168 matrix too. For each element in the imported matrix, there is a value in the factor matrix, which will solve the following problem: $v_{i,j} \cdot x_{i,j} \in [t_{i,j} - tolerance, \ldots, t_{i,j} + tolerance]$ where j is the time index and i the district index. Furthermore, s is the simulated, and t is the imported target value. The tolerance value was included here as it cannot be expected that the calculation will match the exact target value every time. The full pseudocode is shown in algorithm 1.

As shown in the pseudocode, the algorithm uses five matrices. The first is I that contains the imported data. Here, $t_{i,j}$ is denoted by $I[i, j]$ since the imported value is the target value. The second one is X. X holds a factor for each element of I. The values of X can almost directly be inserted into the code from listing 3.3 to recalculate the consumption values. S, the third matrix, consists of a “softness countdown”. If the algorithm doesn't find a valuable factor after three runs, the tolerance will increase by 30%. This matrix keeps the countdown values so that the tolerance can be adjusted when the countdown reaches zero. The fourth matrix, G, consists of the stages that each factor passed or has to pass. If a factor results in a value that is within the tolerance, the stage value gets decreased. If the stage reaches zero, the factor will be assumed as correct. If the result is outside the tolerance, the stage gets reset to three. The next factor then has to pass the stages again. The last matrix, T, handles the individual tolerances for each factor. Then, the UPDATE function gets defined. This method updates the matrices each time it gets invoked. Therefore, this method can easily be integrated into the `ThreadingExtension` from listing 3.2 to get called at every full simulation hour. In this method, the factor of the district with index i and hour j is getting adjusted. This adjustment is based on the formula shown in line 29. Afterward, the new factor will be calculated as the mean value of this factor and the last four factors. The algorithm is considered terminated when no more action is going to be taken. Obviously, this point is reached if the matrix G only consists of zeros.

However, as this algorithm is not trivial anymore, it needs to be proven.

Claim. The algorithm 1 used to adjust the simulation to external data is correct.

Proof. The first property that needs to be proofed here is whether the algorithm terminates. How to easily see, the algorithm will terminate as soon as all elements of matrix G reach zero. As G was initialized with the number three completely, the values must be decreased at any position in the code. The only situation where this happens is in line 27. The only way to reach this line is when the difference between the simulated and the imported value is smaller than the tolerance value of this factor, or the tolerance is higher than the actually imported value. Since the first part of this condition is not predictable due to the arbitrary simulation, the second part is. Assuming the worst-case scenario now, the algorithm will never be able to find a proper factor $x_{i,j}$ for at least one imported data value. In this situation, the tolerance will increase by 30% at every third iteration of the simulation. This is because of the softness countdown matrix used in lines 22 to 25. The tolerance will also not be decreased somewhere else. On the other hand, the imported value does not get changed at any position of the code. The tolerance value thus will be greater than the

Algorithm 1 This algorithm consumes data from external and fits the simulation to it.

```
Require: Matrix $I$ of $7 \times 168$ values to be imported.
X ← new Matrix(7, 168)                          ▷ Factor matrix which will be adapted
S ← new Matrix(7, 168)                          ▷ Softness countdown matrix
G ← new Matrix(7, 168)                          ▷ Stage matrix
T ← new Matrix(7, 168)                          ▷ Tolerance matrix
history ← new Array{-1, -1, -1, -1, -1}         ▷ history with five places for old factors
for $i \leftarrow 1, \dots, 7$ do
    for $j \leftarrow 1, \dots, 168$ do
        $X[i,j] \leftarrow 1.0$                 ▷ Starting with 1.0 for each factor
        $S[i,j] \leftarrow 3$                   ▷ Amount of unsuccessful runs for each factor is three
        $G[i,j] \leftarrow 3$                   ▷ Amount of stages each factor has to pass is three too
        $T[i,j] \leftarrow 1000$                ▷ Initial tolerance value for each factor.
    end for
end for
function UPDATE                                 ▷ This method is supposed to be called hourly to update $X$
    for $i \leftarrow 1, \dots, 7$ do
        for $j \leftarrow 1, \dots, 168$ do
            if $G[i,j] = 0$ then                ▷ If the factor already passed all three stages
                continue                        ▷ it finished and can be left out
            end if
            $v_{i,j} = GetSimulatedValue(i,j)$  ▷ Get the result from simulation
            $difference = I[i,j] - v_{i,j}$     ▷ and calculate the difference
            if $S[i,j] = 0$ then                ▷ If $S$ reached zero at this position
                $T[i,j] \leftarrow T[i,j] + T[i,j] \cdot 0.3$   ▷ the tolerance will be softened
                $S[i,j] \leftarrow 3$
            end if
            if $|difference| \leq T[i,j] \vee T[i,j] > I[i,j]$ then   ▷ If the difference is within
                $G[i,j] \leftarrow G[i,j] - 1$  ▷ the tolerance, the factor passes another stage
            else                                ▷ Otherwise, a new factor must be calculated
                $x_{i,j} \leftarrow \frac{I[i,j]}{v_{i,j}} \cdot X[i,j]$
                $divisor \leftarrow 1$
                $sum \leftarrow x_{i,j}$
                for $k \leftarrow 5, \dots, 2$ do   ▷ The new factor is a mean of the new factor
                    $history[k] \leftarrow history[k-1]$   ▷ and the four calculated before
                    if $history[k] \geq 0$ then
                        $sum \leftarrow sum + history[k]$
                        $divisor \leftarrow divisor + 1$
                    end if
                end for
                $history[1] \leftarrow x_{i,j}$
                $X[i,j] \leftarrow \frac{sum}{divisor}$   ▷ Set the new factor
                $S[i,j] \leftarrow S[i,j] - 1$  ▷ Decrease the softness countdown for this factor
                $G[i,j] \leftarrow 3$           ▷ Reset the stages this new factor must pass
            end if
        end for
    end for
end function
```

imported value sometimes. Therefore, the algorithm will terminate, no matter which values were imported.

Furthermore, the correctness of the results must be shown. These results cannot really be verified because of the unpredictable simulation. The internal mechanisms of Cities:Skylines are unknown and give the impression of calculating somewhat random values. However, it can be proven that the algorithm will try to find the best possible factor $x_{i,j}$ for every i and j, no matter which values are submitted in I. Therefore, given a random matrix I of size seven times 168. After initializing, each run of the UPDATE method will change X's values. Since X is initialized with 1 in every cell, the algorithm will compare the fundamental simulation values with the imported ones at the first run. The algorithm will then compare the simulated value scaled by $x_{i,j}$, with the imported value $I[i,j]$ in all further runs. Based on this, a new factor will be calculated using the following formula:

$$\begin{aligned} x_{i,j} &= \frac{I[i,j]}{v_{i,j}} \cdot X[i,j] \\ &= \frac{I[i,j]}{r_{i,j} \cdot X[i,j]} \cdot X[i,j] \\ &= \frac{I[i,j]}{r_{i,j}} \end{aligned}$$

Let $r_{i,j}$ here be the simulated value, which was not up- or down-scaled by factor $X[i,j]$ from the last iteration. This means that the new factor will be computed by the external data, divided by the value the simulation would initially return. In other words, the new factor indicates the ratio between these two values. This ratio will always scale the next simulation value into the direction of the most current value at the time. Using the mean over the last five calculated factors weakens the ratio of the new factor a bit. Nevertheless, the newly created factor will always lead to $I[i,j]$, based on the most current result from the simulation. From this follows the claim. $\square$

This algorithm will not always perfectly fit the simulation to the imported values. The investigation of this algorithm feed with random and real-world data will be the object of the experiment in chapters 5 and further. Furthermore, the parameters used in this algorithm are not perfectly chosen either. Especially using a lower tolerance, a higher amount of stages, and a lower tolerance adjustment percentage can improve the algorithm's outcome.

However, the development of this mod was not that easy. Apart from the trial-and-error developing strategy mentioned above, it was hard to find the `CommonBuildingAI` class. In fact, this is the only class that ultimately determines the consumption rates of the city. The first version of this modification was based on the `CitizenAI` class. This class implements a method similar to `HandleCommonConsumption`. Though both are applying the same Harmony patch, they do not result in the same. Therefore, it seems that the whole city's electricity consumption gets calculated due to the buildings and not due to their citizens. Moreover, not all parts of the city seem to be adjustable with this kind of adaption. Although the traffic noise emission can be adjusted by applying the patch to all sub-classes of `CarAI`, the traffic density, in contrast, cannot. The citizen density also cannot be adapted with the methods presented in this work. Both of these counterexamples seem to be too complex. Adapting them will probably mean a lot more work to do.

4

Experiment I: How Accurate Is (a Modded) Cities:Skylines?

Before evaluating the simulation of Lübeck injected with the real-world data, an experiment testing the overall validity of the simulation must be conducted. Therefore, the game ran over a defined period, recording some main parameters every hour to validate them afterward. The RealTimeMod takes a special place here, as it significantly changes the simulation to make it more realistic. This experiment shows the overall performance when using Cities:Skylines along with mods as a realistic simulation.

4.1 Test Setup

This first experiment's purpose is to validate an unmodified simulation, which was generated and improved by mods but had no contact with data from reality. In order to achieve this, the test setup only contains the simulation itself with only enough enabled mods to keep the game running correctly. To export the data, the CSExtractDataMod described in section 3.2 was used. Furthermore, parameters that were set during recording can be seen in table 4.1.

Name	Value
Starting date	12th September
Starting population	$\approx 227,000$
Simulation speed	8

Table 4.1: These are the parameters that were set during the first experiment.

The following table 4.2 shows the simulation parameters exported by the CSExtract-DataMod. Thereby, the consumption and emission values were recorded over ten weeks, while the traffic and noise data were only recorded over one week. The difference in the time of recording is based on simply stability causes. A more extended recording period provides more resilient data as single outliers do not weigh as much in one day. On the other hand, a longer recording time will inevitably result in a much bigger data set to be exported. This gets important, especially when it comes to the traffic and noise data. Both parameters are read through the NetManager, which reads the data for each of the more than $23,000$ road sections separately. Exporting such a high amount of values for each hour for more than one week leads to an unstable game software status, which is likely to crash.

Parameter	**Unit**
Electricity consumption	kW
Water consumption	m^3
Heating consumption	kW
Sewage emission	m^3
Garbage emission	not further defined pieces
Traffic density	%
Noise density	%

Table 4.2: This table shows the parameters along with their units that the CSExtractDataMod will export.

The next section will define the testing method to validate the taken experiment using sanity checks and aggregated real-world data.

4.2 Sanity Checks

Validating a complex simulation as the city of Lübeck in multiple dimensions is not trivial regarding the amount of data that needs to be compared. Therefore, so-called "sanity checks" were used to check the overall validity of the simulation's exported data. Those sanity checks listed below are mainly intended to evaluate typical consumption, such as electricity, water, and heating, just as the accumulation of garbage and sewage production. Furthermore, the traffic flow along with the noise emission during the time was sanity checked too.

At the time of writing, no detailed real-world data (such as statistics about the city's consumption values per hour, especially not for the Lübeck city) were available to create valid sanity checks from it. However, the earlier introduced RealTimeMod provides a reasonable basis here: The mod's settings let the user of the simulation decide some basic citizen behavior, such as when they usually wake up, go to work, go to school, go home, or go to bed. Combining these settings with the following common-sense assumptions will result in a handful of valuable sanity checks which can be used to superficial validate the exported data without using detailed real-world data.

A. Citizens consume and emit a minimum when they sleep at night.
B. Citizens start consuming and emitting as soon as they wake up and stop when they go to bed.
C. Citizens constantly consume and emit during the day.
D. Citizens consume more when they are at work as they use machines consuming more electricity or water and emitting more garbage.
E. Citizens usually move through the city when they are awake, especially when they go from home to work/school and vice versa, resulting in a phenomenon named "Rush Hour".

Using these simplified low-level assumptions and combining them with settings from the RealTimeMod leads to the following six sanity checks in table 4.3.

No.	**Assumption**	**Sanity check**
1.	City wakes up at 6 A.M.	Consumption rates, such as electricity and water, along with sewage and garbage emission, will rise significantly. Traffic and noise emissions will not increase.
2.	City goes to sleep at 10 P.M.	Consumption rates, such as electricity and water, along with sewage and garbage emission, will become significantly lower. Traffic and noise emissions will also decrease.
3.	Work start hour at 9 A.M.	Consumption rates, such as electricity and water, along with sewage and garbage emission, will increase significantly. Traffic and noise will have a peak at this time.
4.	Work end hour at 6 P.M.	Consumption rates, such as electricity and water, along with sewage and garbage emission, will become significantly lower. Traffic and noise will have another peak.
5.	School starts at 8 A.M.	Consumption and emitting rates will not change significantly, but traffic and noise will increase at this time.
6.	School ends at 2 P.M.	Consumption and emitting rates will not change significantly, but traffic and noise may have another local peak here.
7.	No work or school at weekends	Sanity checks 3. to 6. do not change anything on Saturday and Sunday.

Table 4.3: Since there is no detailed real-world data of the Lübeck city yet, these sanity checks help validate the overall accuracy of the simulation.

Summarizing the sanity checks during a day will result in a somewhat timetable, starting with a local minimum of consumption/emission and traffic/noise rates at 6 A.M. In the gap between 6 A.M. and 9 A.M., the consumption/emission rates will increase significantly, while the traffic and noise level will reach their first peak. After that, the consumption and emitting of resources will almost stagnate until 6 P.M. when most citizens go home and will arrive at the minimum again at 10 P.M. when most citizens go to bed. Meanwhile, the traffic density and noise emission will face two peaks: the first if the school ends at 2 P.M. and the second at 6 P.M. when work ends.

Even if the exported data cannot be compared hour-by-hour with real-world data directly, the simulated electricity and water consumption actually can be compared with consumption data thankfully provided by the TraveNetz GmbH. This consumption data consists of the overall consumption of an average day, distributed by the administrative district. However, these values can be compared with the exported data. This adds two more sanity checks, which are shown in table 4.4.

No.	Assumption	Sanity check
8.	Electricity consumption	The mean electricity consumption aggregated by 24 hours and administrative district matches the values provided by the TraveNetz GmbH.
9.	Water consumption	The mean water consumption aggregated by 24 hours and administrative district matches the values provided by the TraveNetz GmbH.

Table 4.4: These two sanity checks were added to countercheck the electricity and water consumption with the data provided by the TraveNetz GmbH.

The following section will apply these sanity checks to the exported data of this experiment.

4.3 Results

The exported results of this experiment can be seen in the graphs of figure 4.5 and table 4.6. Creating an export over ten weeks of consumption values respectively one week of traffic and noise data results in a very high amount of data. Therefore, the graphs of figure 4.5 show average day values. These are divided into two different types of days. The first one is the average weekday indicated by the blue graphs. Secondly, the weekend day values are shown as the red graphs on the right side. Since RealTimeMod's settings apply to the whole city, the values used in figure 4.5 were also accumulated by Lübeck. The traffic and noise values of all streets in the town were summed up too. The dashed lines within each graph are indicate the timings of the sanity checks of table 4.3.

General Consumption Data

The graphs 4.5a to 4.5d show the average consumption values of Lübeck city.

1. The first check is fulfilled by each of the five graphs. Starting from 6 A.M., the consumption and emission values are leaving their global minimum from the night.
2. In contrast, the second sanity check is not directly fulfilled by any of the five graphs. The consumption values are not changing at 10 P.M. However, the consumption rates are still going down two hours later, between 12 A.M. and 1 A.M.
3. The third check is indicated by the most significant change. Starting from 9 A.M., the consumption values are reaching their global height.
4. The fourth check shows another contrast to the previous one. At 6 P.M., nothing changes, so this sanity check is not met.
5. The fifth one is somewhat fulfilled. Between the times 8 A.M. and 9 A.M. a slight consumption increase can be seen.
6. The sixth check is fulfilled. Around 2 P.M., none of the consumption values changes.

7. Sanity check seven is not fulfilled at all, as there are no differences between the weekday and weekend day graphs.
8. and 9. are not fulfilled at all. Taking a look into table 4.6 reveals that there is a high difference between the simulation's electricity respectively water consumption and the consumption rates provided by the TraveNetz GmbH.

Traffic and Noise

The situation becomes a little more complex when it comes to traffic and noise data. As can be seen in graphs 4.5e and 4.5f, the curves are not much correlating anymore.

First, none of the sanity checks one to six are fulfilled by the weekday graph 4.5e. As shown there, the traffic density reaches its all-time high at 1 A.M. After that, it constantly decreases until 12 A.M. The weekend traffic density accumulation is neither fulfilling one of the sanity checks, apart from check seven. For some reason, the traffic density here reaches its all-time high in the morning between 3 A.M. and 5 A.M.

The traffic noise emission in graph 4.5f does not look much different. This chart does not fulfill almost any of the sanity checks from table 4.3. In the case of the weekday graph on the left side, the density also reaches its highest point at 3 A.M. Afterward, it decreases until 9. At this point, the noise emission increases a bit, which can be seen as a fulfillment of check three. After reaching the global minimum at 3 P.M., it starts to increase until 9 P.M. This can be seen as another slight fulfillment of checks two and four. In contrast, the weekend graph on the right side is entirely different. Starting from 12 A.M., the course is constantly increasing until noon. Afterward, it constantly decreases until 11 P.M. Therefore, sanity check seven is definitely fulfilled. Moreover, check one and two also can be seen as somewhat meet, because the course increases at 6 A.M. and decreases at 10 P.M.

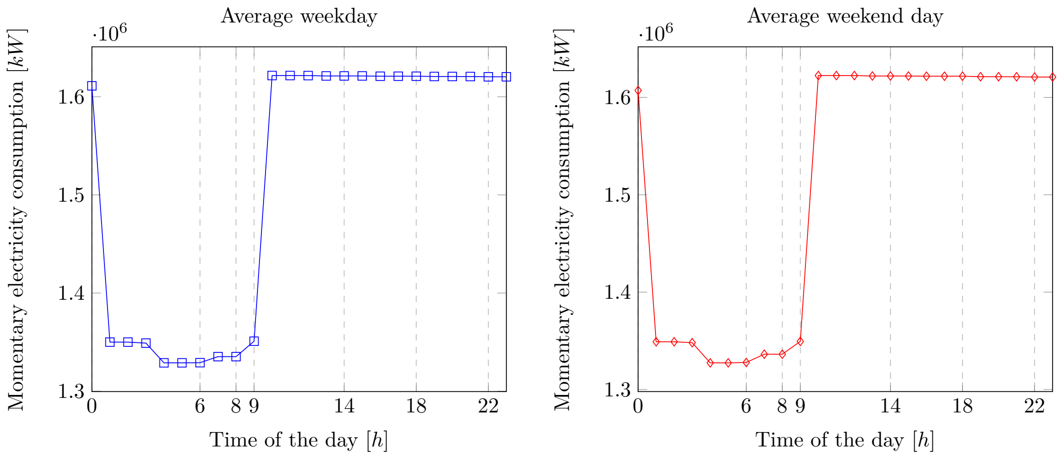
4.4 Interpretation

As can be seen in the results, not all sanity checks were passed by the exported data. In fact, regarding the consumption and emission rates, the exported data fulfilled just four of the nine sanity checks. Moreover, the simulation did not match TraveNetz's water and electricity consumption, which would otherwise have been very surprising. A comparison of these data can also be seen later in experiment III, figure 6.3b.

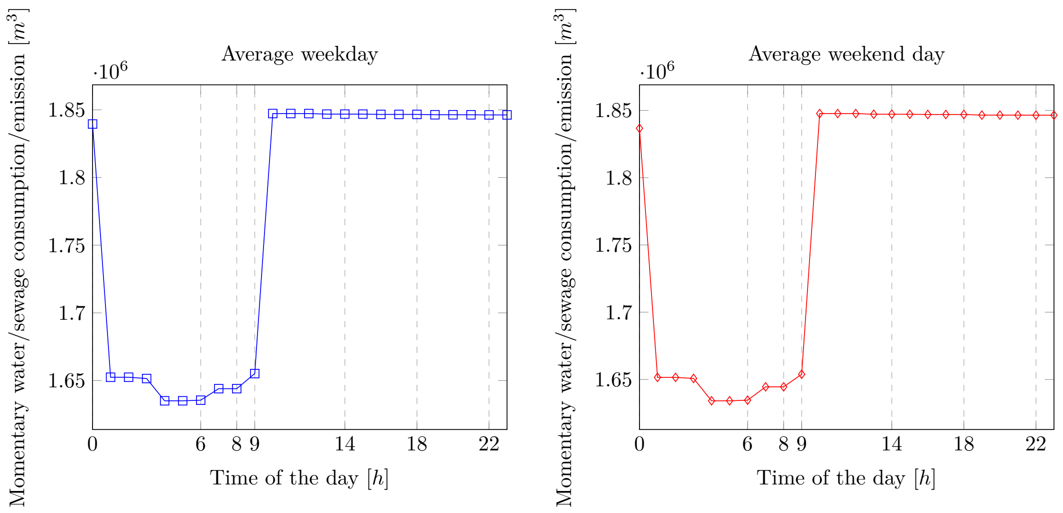
The consumption and emission charts look promising, nevertheless. All of these charts had a clear timetable with global consumption and emission minimum at night. Furthermore, all of them showed global height during the day, which starts at 8 respectively 9 A.M when the citizens are expected to go to work. On the other hand, this timetable also gets applied to the weekend days. This behavior contradicts RealTimeMod's promise of adding weekends to citizen's behavior (see section 2.3).

The traffic and noise graphs reveal a significantly worse image. Almost none of the sanity checks can be applied to the exported charts. Apart from the weekend graph from the noise export 4.5f, all charts have at least one peak before 6 A.M. for no apparent reason. It seems that some kind of bug could be responsible for these results.

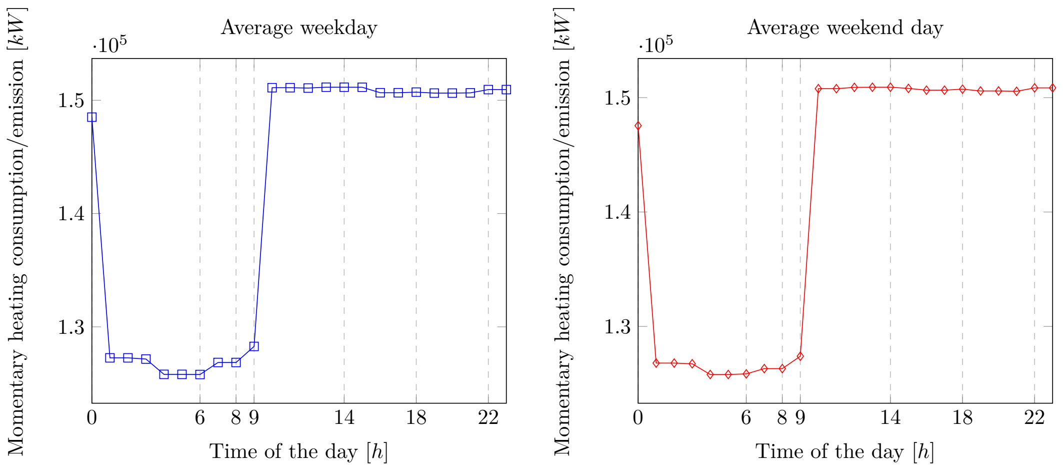
On the whole, the results from the sanity checks leave a mixed image. It is essential to highlight here that these sanity checks aren't the most accurate evaluation method. They are



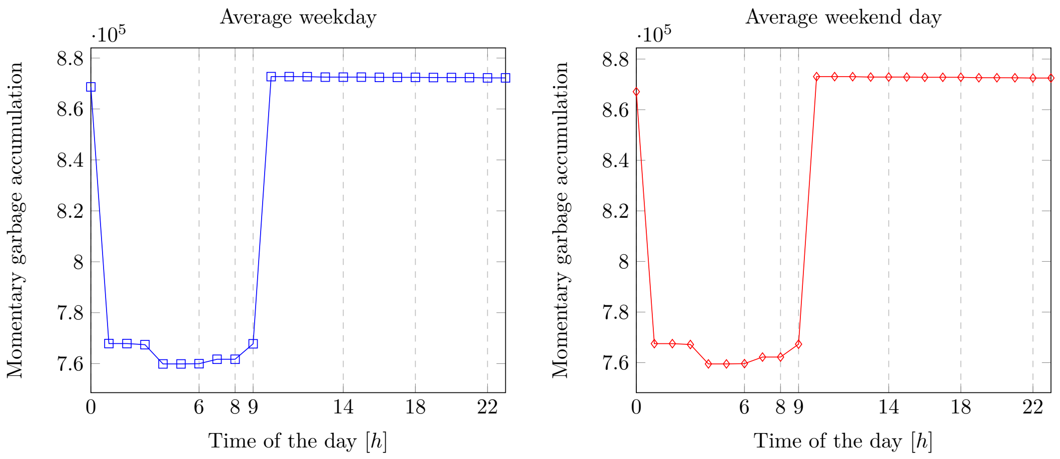
(a) Electricity consumption of Lübeck city.



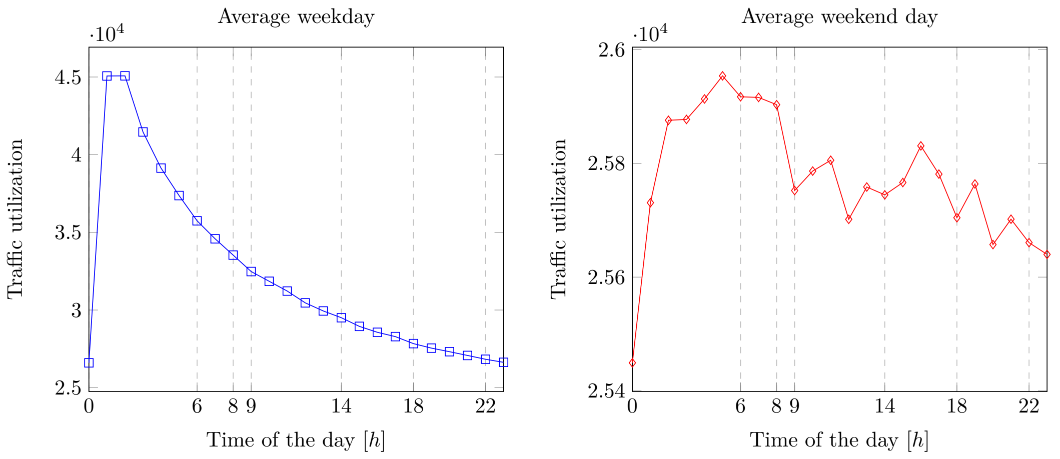
(b) Water/sewage consumption/emission of Lübeck city.



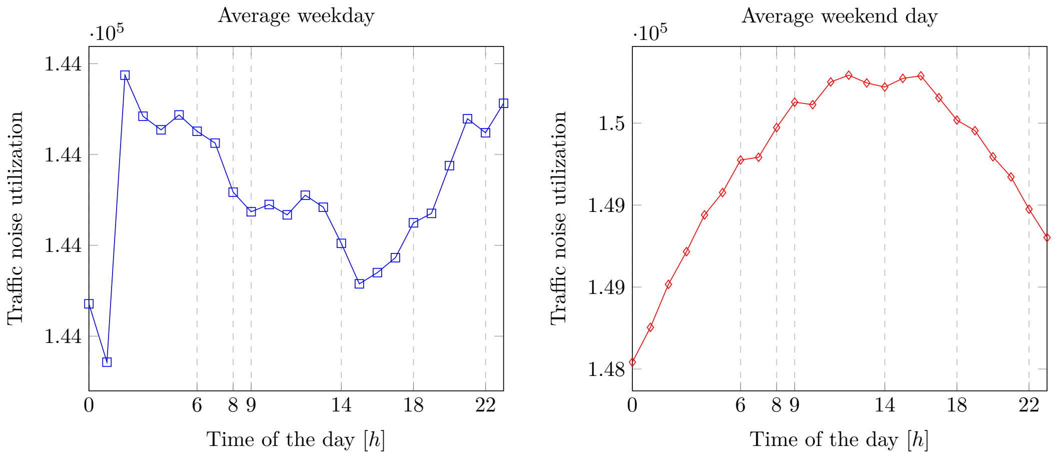
(c) Heating consumption of Lübeck city.



(d) Garbage emission of Lübeck city.



(e) Accumulated traffic density of Lübeck city.



(f) Accumulated traffic noise density of Lübeck city.

Figure 4.5: These are the consumption and emission rates during a period of ten weeks. Since ten weeks of hourly recording results in a high amount of data, the graphs here only show the accumulated values for the whole Lübeck city. The blue charts on the left side present the consumption/emission during an average weekday, the red graphs during an average weekend day. The vertical dashed lines in the graphs indicate the sanity checks from table 4.3.

.

District	**TraveNetz's electricity** [kWh]	**Simulation's electricity** [kWh]
Innenstadt	10974	61217.4952380952
Moisling	5331	26632.7238095238
St. Gertrud	16623	343707.742857143
St. Jürgen	20565	424223.847619048
St. Lorenz	26190	432836.504761905
District	**TraveNetz's water** [m^3/h]	**Simulation's water** [m^3/h]
Innenstadt	171	55055.4380952381
Moisling	67	34386.0285714286
St. Gertrud	224	456051.019047619
St. Jürgen	292	460588.628571428
St. Lorenz	353	511396.666666667

Table 4.6: This table shows the comparison between the water and electricity values provided by TraveNetz in contrast to the simulation's values.

neither based on studies nor scientific evaluations but intuitions and assumptions. However, since most data runs through or is generated by the open-source and well-documented RealTimeMod, it should not be much effort to adjust the logic behind it. This leaves space for improvements. In summary, Cities:Skylines is, in some points, somewhat accurate, in some points not. Creating a simulation fulfilling all of the sanity checks used here should be an object of further investigation.

5

Experiment II: How Accurate Is an Adapted Cities:Skylines?

The last experiment validated the overall simulation. However, it has used the CSExctractDataMod (see section 3.2), but none of the data were used to be adjusted by the CSImportDataMod (section 3.3). That will change in this experiment. The purpose of this experiment will be primarily the validation and evaluation of the CSImportDataMod. Therefore, an amount of randomly generated numbers will be used as an input. Afterward, the fitting between the simulation and the random numbers will be evaluated. If it is possible to feed the simulation with random data, it will show that it can be adapted using almost any real-world data.

As this experiment is meant to confirm the proof-of-concept of the CSImportDataMod, it only focuses on one of the parameters listed in section 3.2. The adjustment of all the other parameters works in the same way; it does not matter which one is chosen. However, the electricity consumption will be adapted in this experiment.

5.1 Test Setup

The setup of this experiment is almost similar to the one before. The simulation will be started in the same way but with an activated CSImportDataMod. This means that the adaption phase will begin as soon as the simulation reaches Sunday midnight the first time. After the CSImportDataMod finished its calculations (all elements of G were set to zero), the CSExtractDataMod was used to export the data for one week. After that, this data will be used for the evaluation in section 5.2. The parameters set during the simulation time are shown in table 5.1.

As written above, the primary purpose of this experiment is to evaluate how well the simulation is fittable to random data. Therefore, a matrix of seven times 168 random values will be submitted to the CSImportData mod. This dataset covers all seven districts and a week of hourly data.

Name	**Value**
Starting date	12th September
Starting population	$\approx 227,000$
Simulation speed	8
Tolerance	1000
Amount of stages that must be passed	3
Amount of softness stages	3
Softness factor	0.5

Table 5.1: These are the parameters that were used by the simulation in the second and third experiments. In contrast to the previous experiment's setup (table 4.1), the CSImportDataMod brings some adjustable values too, which are also listed here.

5.2 Results

In order to evaluate this fitting, the week after the adjustments by the CSImportDataMod finished were recorded and exported using the CSExtractDataMod. The graphical processing for each district can be seen in figure 5.3. This figure shows seven graphs, one for each district of the city. Here, the x-axis represents the hours h from 0 to 167, while the y-axis represents the momentary electricity consumption in kW. The blue line is the momentary electricity consumption generated by the simulation. In contrast, the red dots show the imported random values. Therefore, the outcome of this experiment is better if the blue line fits closer to the red dots.

However, only evaluating the simulation with the visual interpretation of the graphs is somewhat inaccurate. Therefore, the Root-Mean-Square-Error (RMSE) was chosen as the error evaluation. With v_i denote the simulated value, t_i the random target value, and n the size of the data set, calculating the error ϵ using the RMSE is now defined as follows [8]:

$$\epsilon_{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(t_i - v_i)^2}{n}}$$

In short, the RMSE calculates the mean Euclidean distance between each relating simulated and imported value. Furthermore, the RMSE is scale-dependent. Using the scale-independent variant of the RMSE algorithm (namely the Root-Mean-Square-Percentage-Error) is unusable here, as it results in infinity when any of the values is zero [10]. Unfortunately, this happens in some v_i values.

Another value that can be calculated to evaluate the performance is the Mean-Absolute-Error (MAE). The MAE value is mainly the overall mean distances between the imported and the simulated value. The formula is denoted as follows [8]:

$$\epsilon_{MAE} = \frac{1}{n}\left(\sum_{i=1}^{n}|t_i - v_i|\right)$$

In contrast to the RMSE, the MAE has a lower vulnerability due to outliers, as the error does not get squared here. However, just as in RMSPE's case, the scale-independent variant of the MAE (namely the Mean-Absolute-Percentage-Error) cannot be calculated here, as some of the v_i values equal in zero.

Additionally, the error values of each district will not be totaled to an overall error here. This is because of the location-based import. As described in section 3.3, the data gets imported separately for each district of the city. Consequently, the fitted data has to be also compared for each district separately then. Another reason is that the error values differ a lot from district to district. Totaling them up would mean a loss of interpretation room, therefore.

Table 5.2 shows the ϵ_{RMSE} and ϵ_{MAE} values for each district, along with the difference Δ of both ϵ. As the RMSE is more sensitive to outliers, a higher difference between ϵ_{RMSE} and ϵ_{MAE} means a higher impact due to outliers in the error calculation. The interpretation of these values will be the object of the next section.

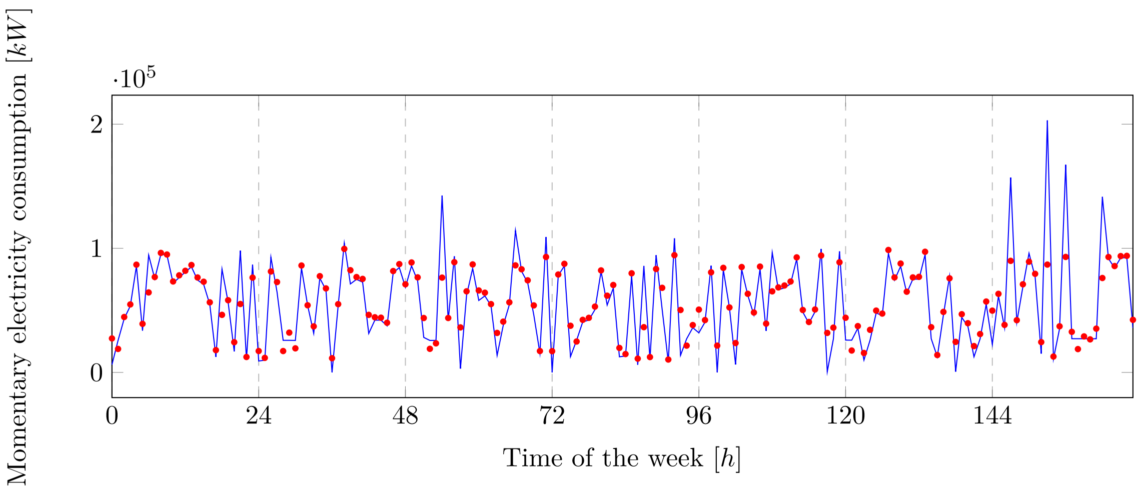
District	ϵ_{RMSE} $[kW]$	ϵ_{MAE} $[kW]$	Δ $[kW]$
St. Lorenz Nord	60886.18	21374.77	39511.40
Buntekuh	1251388.06	115325.33	1136062.73
Moisling	17642.56	8770.46	8872.1
St. Lorenz Süd	189166.53	26446.98	162719.55
St. Jürgen	44365915.25	4537464.08	39828451.17
St. Gertrud	106374955.06	8231442.57	98143512.49
Innenstadt	17843007.97	1389936.08	16453071.9

Table 5.2: This table shows the Root-Mean-Square-Error (RMSE) and Mean-Absolute-Error (MAE) of the simulated data for each district. The data on which these errors are based can be seen in figure 5.3. Moreover, the far-right column Δ shows the difference between ϵ_{RMSE} and ϵ_{MAE}. As the RMSE is more sensible to outliers, a higher difference indicates a higher impact due to outliers. All values are rounded to the second decimal place.

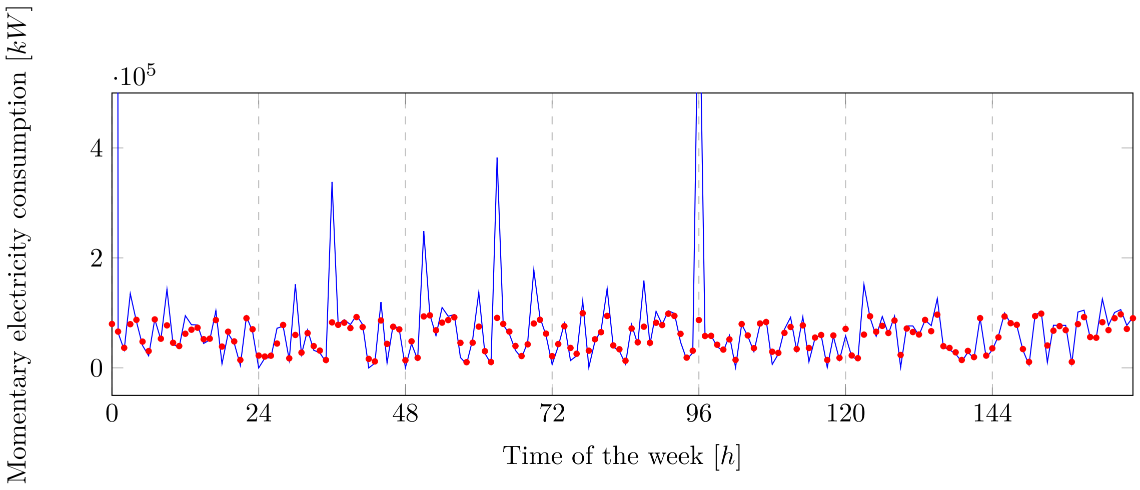
5.3 Interpretation

Interpreting the result reveals a mixed picture. On the one hand are the graphs in figure 5.3. Apart from some heavy outliers, the blue "simulated" line seemingly fits the red dots quite well for every district. On the other hand are the RMSE and MAE values in table 5.2. Since these are calculated as the mean distance between two relating dates, a lower value attests to a better fitting. However, the ϵ_{RMSE} and ϵ_{MAE} value is for every district very high, although it is not equally high for all.

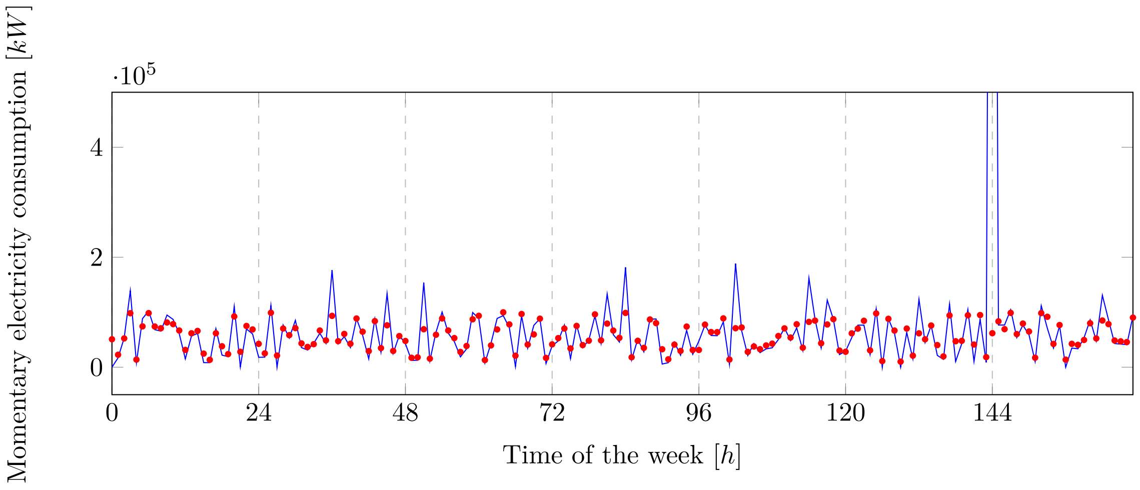
The district with the lowest ϵ_{RMSE} respectively ϵ_{MAE} value is Moisling. Apparently, this district is also the only one having no outliers above 500,000 kW. Moreover, the difference Δ between ϵ_{RMSE} and ϵ_{MAE} is the lowest too, which indicates the most negligible impact due to outliers. Nevertheless, both errors are higher than 150% of the lowest value from the imported data, with $\approx 17,642.56$. On the other side, the district with the highest ϵ_{RMSE} and ϵ_{MAE} value is St. Gertrud. This district also simulated the highest outlier of the whole set, consuming an enormous amount of 1,378,839,088 kW at 12 A.M. on Saturday.



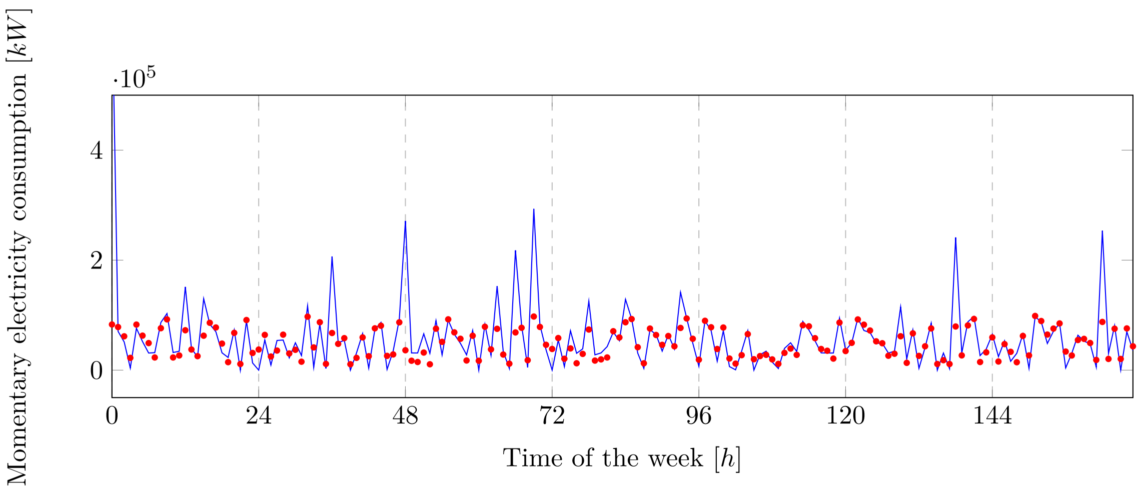
(a) Electricity consumption of Moisling by hour



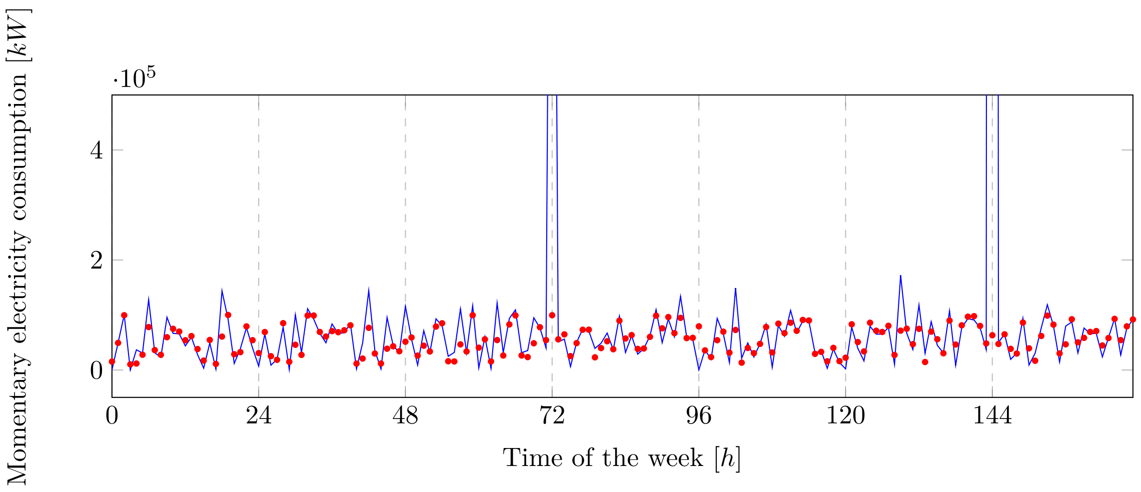
(b) Electricity consumption of Buntekuh by hour



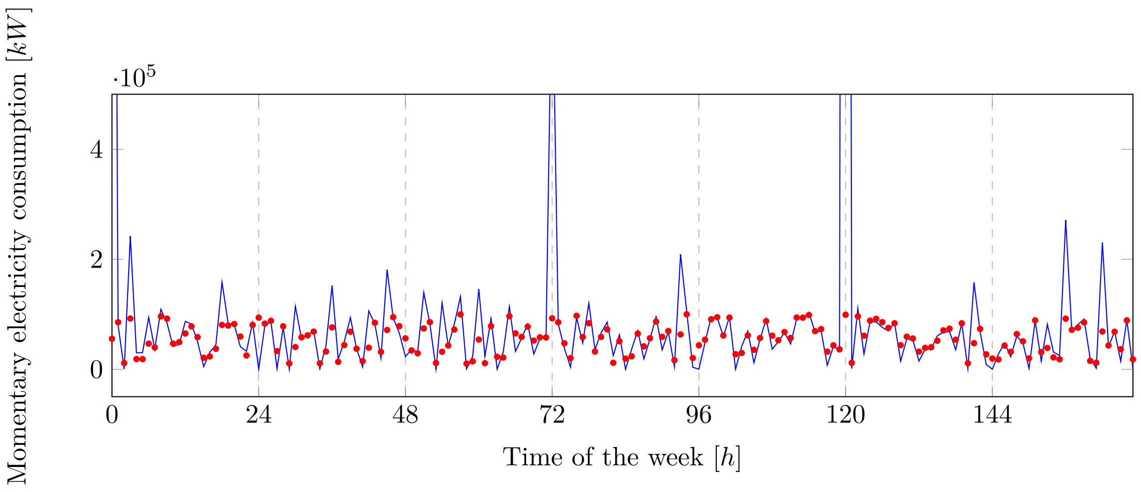
(c) Electricity consumption of St. Lorenz Süd by hour



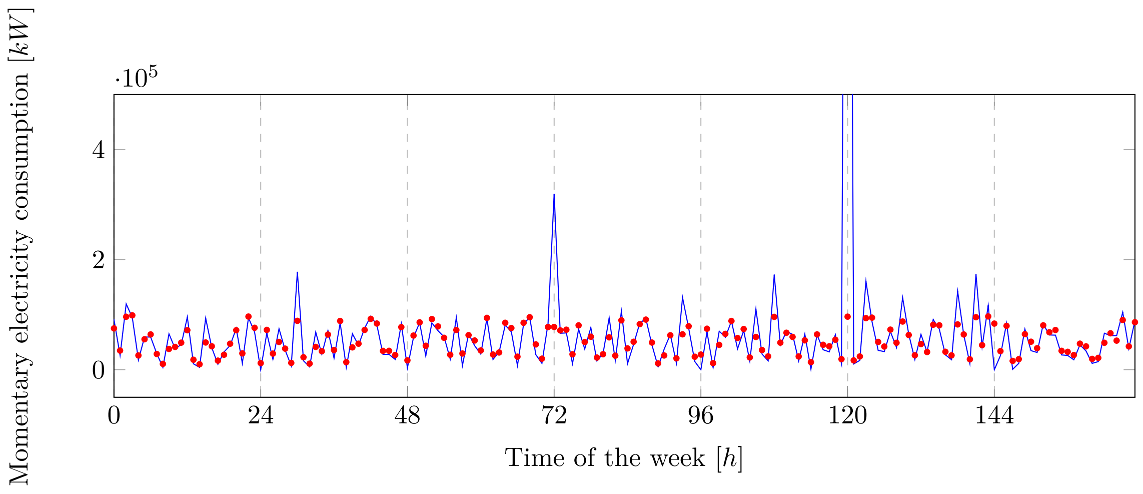
(d) Electricity consumption of St. Lorenz Nord by hour



(e) Electricity consumption of St. Gertrud by hour



(f) Electricity consumption of St. Jürgen by hour



(g) Electricity consumption of Innenstadt by hour

Figure 5.3: Electricity consumption of the first week after the CSImportDataMod finished adapting the city. Each graph shows one of the seven districts. The blue line indicates the simulated values. The red dots represent the random values the simulation should be fitted to. Since some of the values seem to converge to infinity, these graphs are cropped at $5 \cdot 10^5 kW$.

This value is more than $21,813$ times higher than the district is supposed to have at this time. The difference Δ is the highest here, too, confirming the highest impact from outliers.

Characterizing the outliers is not trivial. In this part, the focus will lay on the upper outlier as these seem to be much more decisive than the lower ones. Therefore, the higher outliers have a more significant impact on the error calculation from the last section. However, analyzing the lower outliers will go into much more detail and could be a topic for further research. The following table 5.4 shows the highest value for each district. Apart from Moisling, each district's highest value occurs at midnight, which also correlates with the observations from figure 5.3. This expression is confirmed when taking a deeper look at the electricity consumption fitting of St. Gertrud in graph 5.3e. This graph shows two outliers above 500,000 kW. The first one occurs at recording hour 72, the second one at

District	**Highest value** $[kW]$	**Simulation time**	**Day of week**
St. Lorenz Nord	701408	12 A.M.	Sunday
Buntekuh	116277808	12 A.M.	Sunday
Moisling	203072	9 A.M.	Saturday
St. Lorenz Süd	2498432	12 A.M.	Saturday
St. Jürgen	525922416	12 A.M.	Friday
St. Gertrud	1378839088	12 A.M.	Saturday
Innenstadt	231368032	12 A.M.	Friday

Table 5.4: This table summarizes the highest recorded values for each district and the time of their occurrence.

hour 144. As the CSExtractDataMod is programmed to start every recording at 12 A.M. on Sunday, both outliers must have happened at midnight too.

However, one last test can be taken in order to show that the outliers are primarily responsible for the high error values of table 5.4. Therefore, the 90th percentile was determined for every exported district array. This percentile was calculated using the "numpy" Python library [13]. Afterward, the RMSE and MAE values were recalculated as ϵ'_{RMSE} and ϵ'_{MAE}. Thereby, each value higher than the 90th percentile gets excluded from the calculation. This leads to the following table 5.5:

District	**90th percentile** $[kW]$	ϵ'_{RMSE} $[kW]$	ϵ'_{MAE} $[kW]$	Δ' $[kW]$
St. Lorenz Nord	97211.2	12283.74	9644.50	2639.24
Buntekuh	112785.6	10408.1	7056.75	3351.35
Moisling	94622.4	10254.27	6097.83	4156.44
St. Lorenz Süd	105024	10400.57	7821.77	2578.8
St. Jürgen	113787.2	16352.68	10725.4	5627.27
St. Gertrud	108561.6	16181.27	10894.32	5286.95
Innenstadt	95099.2	12981.44	9011.82	3969.62

Table 5.5: The data shown here are the recalculated ϵ'_{RMSE}, ϵ'_{MAE}, and Δ' values. Thus, all values higher than the 90th percentile were excluded from the calculation. As can be seen here, all of the values are lower than the same values from table 5.2. Since both ϵ' errors are closer together now (Δ' is considerably lower than Δ), the impact due to outliers was significantly reduced. All values are rounded to the second decimal place.

As can be seen here, ϵ'_{RMSE} and ϵ'_{MAE} are significantly lower than the relating ϵ values. In the case of RMSE, all of the results are within the range of $10,000$ to $20,000$ kW now. The mean errors are currently in a room of about $6,000$ to $11,000$ kW, leading to a distance Δ' not exceeding $6,000$ kW. This reduced difference indicates a significantly lower outlier impact. Looking closer at the district St. Gertrud, which had the highest errors according to table 5.2, its value is reduced considerably. In fact, ϵ'_{RMSE} is $\approx$ 16181.27 times lower than the ϵ_{RMSE}. The ϵ_{MAE} was decreased by a factor of $\approx$ 755.57. Moreover, St. Gertrud's Δ' is no longer the highest value, attesting that the outlier impact is now in range with the other districts. It is easy to see that removing the highest outliers decreased all errors by a lot, finally.

However, these outliers must be caused by the CSImportDataMod, as they did not occur in the results of experiment I. Furthermore, excluding all values higher than the 90th percentile from the error calculation is not really a deal because the cause of these outliers has not been sufficiently clarified. Researches in the future maybe are adapting this algorithm to lower the error values to somewhat acceptable. All in all, an adapted Cities:Skylines is somewhat accurate, leaving room for improvements.

6

Experiment III: How Close Can Cities:Skylines Be Made to Reality?

The third and final experiment will test how well real-world data can be imported into the existing simulation of Lübeck. Therefore, the same data as in the first experiment were used to adapt Cities:Skylines to it. These dates are aggregated values for an average day that the TraveNetz GmbH thankfully provided. Although they provided water and electricity consumption rates, this experiment will only adapt the electricity consumption, just as in the previous experiment.

Similar to the last experiment, the city was adapted using the CSImportDataMod. After this mod finished adapting the game, the CSExtractDataMod recorded one week of electricity consumption. This exported data was then compared with the imported data.

6.1 Test Setup

In order to conduct this experiment, the CSImportDataMod needs to be adjusted slightly. As the external data are aggregated values here, algorithm 1 needs to average them up before comparing them with the imported data in line 26. Thus, the CSImportDataMod will record each district for one week, sum the electricity consumption up, divide the result by 168, and then compare it with the imported matrix's relating value. The advantage of averaging one week is that both weekdays and weekend days are included in the calculation.

The rest of the experiment is similar to the last one in chapter 5. The parameters used in the simulation can be seen in table 5.1, therefore.

6.2 Results

After adapting the city and exporting one week of electricity consumption directly after the adapting phase ended, the results can be seen in table 6.1.

However, the RMSE and MAE values taking place in the results of the last experiment (see section 5.2) were also calculated here and can be seen in table 6.2.

In addition to this, the Mean-Absolute-Percentage-Error(MAPE) can also be calculated here. In contrast to the MAE, the MAPE is scale-independent (as it results in a percentage

District	**Imported value** $[kW]$	**Average exported value** $[kW]$
Moisling	5331	4434.86
St. Lorenz	26190	29797.24
St. Gertrud	16623	16306.38
St. Jürgen	20565	24163.43
Innenstadt	10974	10613.05

Table 6.1: These are the results of the third experiment. Here, the values in the middle column were imported into the game. The results of the simulation can now be seen in the far right column. In contrast to the second experiment, fewer districts were included. The data from the TraveNetz GmbH cause this. First, there is no data for the district "Buntekuh" yet, and second, the district St. Lorenz Nord and St. Lorenz Süd are accumulated. All values are rounded to the second decimal place.

District	ϵ''_{RMSE} $[kW]$	ϵ''_{MAE} $[kW]$	Δ'' $[kW]$	ϵ''_{MAPE} $[\%]$
Lübeck	2323.55	1755.88	567.68	10.66

Table 6.2: The RMSE, MAE, difference Δ'', and MAPE values shown here are the overall errors based on the results of table 6.1. All values are rounded to the second decimal place.

value), making it comparable with different algorithms. However, no v_i values are evaluated to zero in this experiment. This was the reason for not calculating the MAPE in the last experiment, as this would have resulted in infinity values. The formula of the MAPE is as follows [8]:

$$\epsilon''_{MAPE} = \frac{1}{n}\left(\sum_{i=0}^{n}\left|\frac{t_i - v_i}{v_i}\right|\right)$$

After inserting the parameters, the ϵ''_{MAPE} value of this experiment is $0.1065501354952318 \approx 10.66\%$.

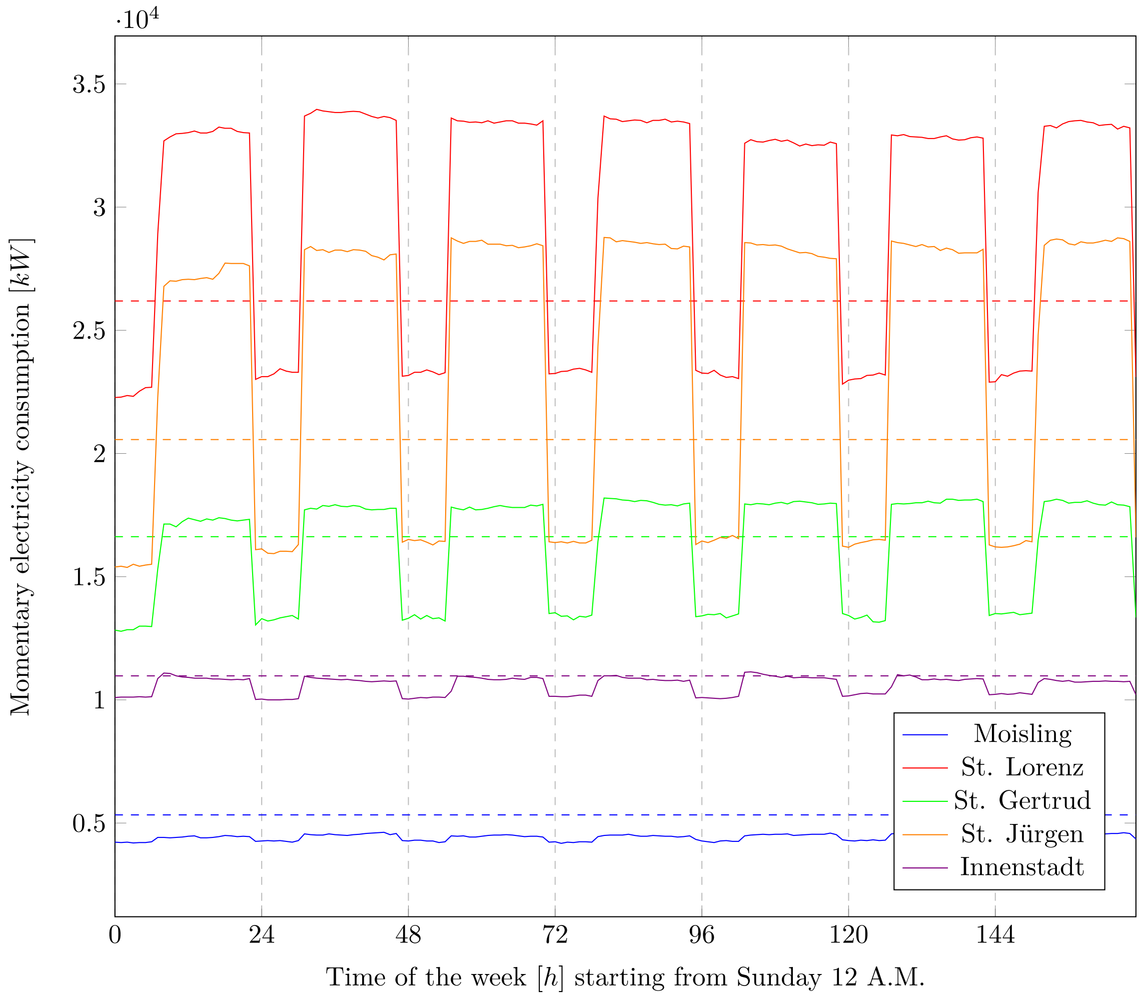
6.3 Interpretation

Taking a closer look at the results in table 6.1 of this experiment reveals a much better overall performance of the CSImportDataMod algorithm. The exported values from the adapted simulation seem to be reasonably close to the imported ones, having a maximum difference of about 3598.43 kW in St. Jürgen.

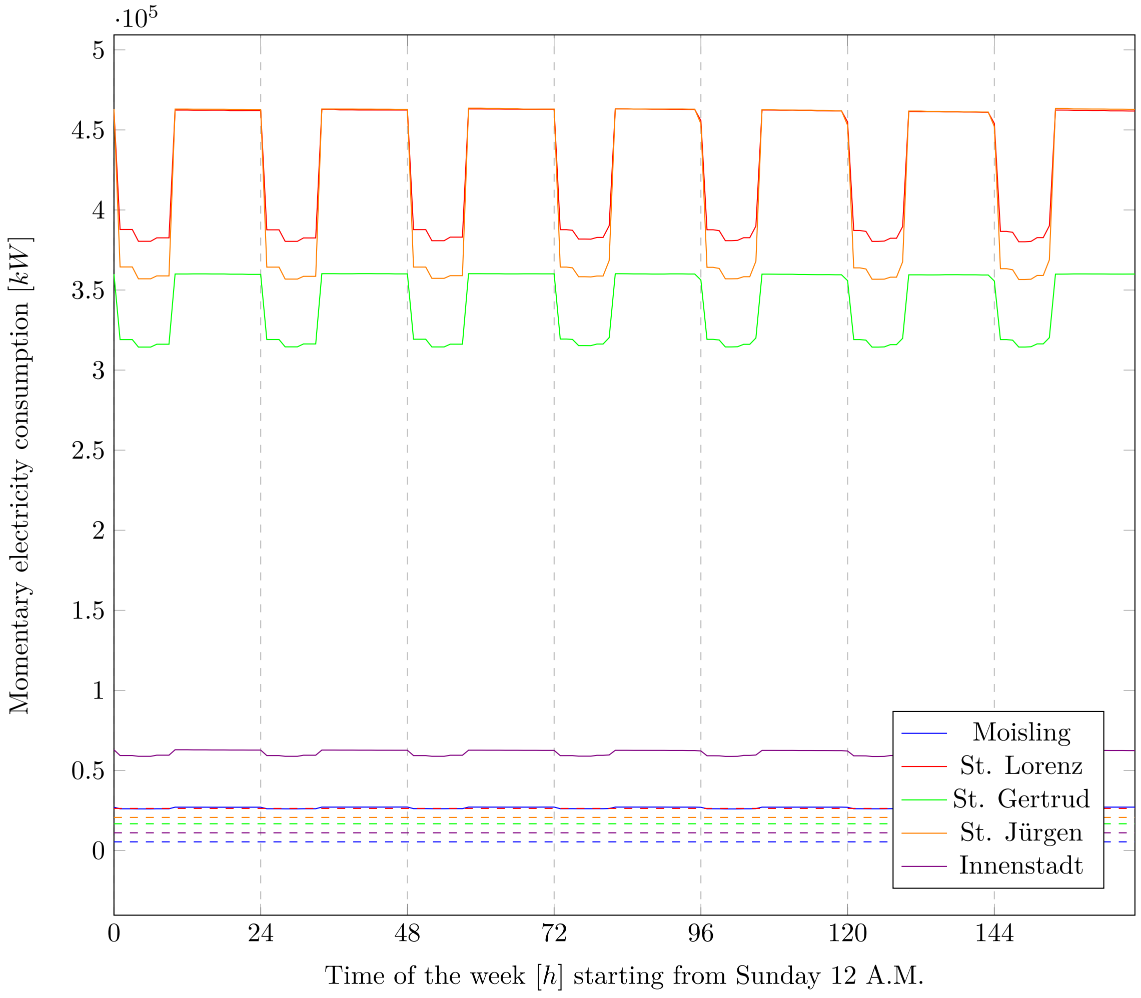
Now, given the ϵ_{RMSE} and ϵ_{MAE} error values from table 5.2 of the previous experiment. Comparing them with the error values ϵ''_{RMSE} of ≈ 2323.55 and $\epsilon''_{MAE} \approx 1755.88$ from this experiment shows significantly lower ϵ''_{RMSE} respectively ϵ''_{MAE} error values. Actually, ϵ''_{RMSE} is more than seven times lower than the lowest ϵ_{RMSE} from table 5.2. Similar applies to ϵ_{MAE}, where ϵ''_{MAE} is almost five times lower than its lowest pedant from table 5.2.

It is striking that the recording this time does not seem to have notable outliers, although it uses almost the same CSImportDataMod as in experiment II. Figure 6.3 gives an overview of the exported electricity consumption used to calculate the average exported value in table 6.1. In this figure, two graphs are shown. The upper one (figure 6.3a) presents the electricity consumption exported for this experiment (continuous line) compared to the average electricity consumption provided by TraveNetz (dashed line). Each color belongs to one district. Since the adapted CSImportDataMod tries to fit the average consumption values to the imported ones, the dashed lines need to be somewhat in the middle of the related continuous line. However, the charts did not show as significant outliers as in figure 5.3 this time. In contrast, the lower graph 6.3b presents the electricity consumption collected simultaneously without an activated CSImportDataMod, compared to the average values from TraveNetz.

All in all, the simulation can fit quite well to TraveNetz's real-world data. Furthermore, the scale-independent ϵ''_{MAPE} seems relatively low, with a percentage error of about 10.66%, and the lower distance Δ'' (compared to experiment II) attests to a relatively slight impact due to outliers. However, future evaluations and collaboration with TraveNetz will show how good these values actually are.



(a) Exported electricity consumption of an adapted simulation by district.



(b) Exported electricity consumption of an unmodified simulation by district.

Figure 6.3: Both graphs show the exported momentary electricity consumption distributed by district. In both charts, each color stands for a different district. While the continuous lines present the simulated values, the dashed lines are the average values for the related district provided by TraveNetz. The upper graph 6.3a shows the exported values from the third experiment using the modified CSImportDataMod. The second chart, 6.3b, shows the data exported simultaneously but without modifying the simulation, in contrast to TraveNetz's data. A remarkable detail here is that the course of both graphs looks very similar.

7

Conclusion

Smart city simulations can help to plan, maintain, and improve urban infrastructure. The computer game Cities:Skylines might help to create a valuable smart city simulation with respect to individual data protection by providing a good simulation environment. This thesis aimed to investigate the game's logic and improve the environment to help achieve this goal.

7.1 Discussion

The results of this work revealed a very mixed impression of Cities:Skylines in general. Therefore, the question arises whether this game, in particular, is a good choice for a valuable smart city environment, especially with the privacy focus.

Cities:Skylines is and was designed as a computer game. The game's creator developed the software primarily to entertain their customers. As described in section 3.2, the simulation holds much more detailed information than communicated through the game's GUI. This information can only be extracted by using modifications.This points out another main problem: Many mods need to be downloaded and installed to make the Cities:Skylines somewhat usable for smart city simulation. Each mod makes the game more unstable. Furthermore, a modification can be a security risk because not all mods are open source and as well documented as the RealTimeMod from section 2.3. Moreover, the community mainly consists of gamers who want to improve their favorite game without creating the most stable, realistic simulation. This impression is confirmed when taking a look at the results of the first experiment (namely, figure 4.5). Although RealTimeMod's improved citizen timetable passed some sanity checks, the related traffic charts did not. It is possible that the hourly traffic is just not as important for the gameplay and so not improved by the gaming community yet. It is also imaginable that the traffic is just buggy due to the high amount of installed mods. During this work, game crashes and bugs due to modifications significantly impacted the workflow with this simulation. Another negative workflow impact was the overall modding process. If the modification is supposed to take over more complex tasks outside the official modding API's range, things are getting complicated quickly. Since the game code is closed source, it is unreadable and not officially documented at all. Developing mods under these circumstances is very hard and follows the trial-and-error principle. Although the second experiment attested general adaptability of Cities:Skylines, the outliers of figure 5.2 indicated a certain limit here. At some point, the simulation does not follow the adaption anymore, and it is incomprehensible why this happened.

However, using Cities:Skylines not only has disadvantages. One of the main advantages is the standalone simulation itself. Cities:Skylines consists of a remarkable high level of detail. Furthermore, the simulation's adaptability due to modifications is an advantage, too, although the modding process is not as easy. The game can be adapted, maintained, or even bug-fixed just with the usage of mods. This makes Cities:Skylines even more usable. Experiment III showed that the adaption of Cities:Skylines could work out quite well. From the privacy point of view, adapting an existing simulation works out better than creating a completely new one based on real-world data. Especially the privacy aspect is denoted by experiment III, too. As can be seen in the results section 6.2, Cities:Skylines can be adapted quite well with a relatively small database of seven numerics in this case. As these real-world numbers are aggregated mean values, there already anonymized. The overall performance does not seem to suffer from it: The course of the districts in both graphs seem to be very similar. This means that the adaption works out without losing the properties pointed out in experiment I.

Using Cities:Skylines as a smart city simulation is very challenging. Nevertheless, this work proved the concept of adapting the game to real-world data. Furthermore, it showed that it could be used indeed to create a smart city simulation with a remarkable low data basis, meaning consideration of individuals' data privacy. Future research may now improve the results.

7.2 Future Work

However, there is still a long way to go. The experiment done in this work revealed some problems which need to be solved in the future. For instance, the traffic density and traffic noise emission were not passed by the sanity checks. Furthermore, the experiment in chapter 5 showed some heavy outliers in the electricity consumption modified by the CSImportDataMod.

The simulation's save game itself lacks several services. For example, Lübeck city does not have any bus lines or bike lanes implemented yet. Furthermore, the city does not consist of any hospitals, police stations, or fire departments. As each of these services probably impacts traffic and noise density, including them in the simulation is highly recommended. This inclusion could be done by extending the GeoSkylines mod [15] to read the desired data from OpenStreetMap as well.

The simulation's electricity and water infrastructures are slightly primitive in the current state. In fact, Cities:Skylines implements power lines and water/sewage pipes out of the box, but mods disabled these. Nevertheless, both power lines and pipes do not have any capacity parameter implemented yet. Furthermore, the simulation also does not implement a deep distribution system for the electricity or water supply chain. Game modifications could add both capacities and a deep supply chain.

Finally, algorithm 1 developed in the context of this work could be improved. As written in section 3.3, algorithm 1 is meant to be a proof-of-concept. Using a more detailed machine-learning mechanism can probably enhance the results of experiments II and III by a lot. Moreover, the adaption used in CSImportDataMod, as seen in listing 3.3, could be improved to include more parameters. For example, the size of the building, the age of the residents, or predefined usage patterns can be inserted here. This improvement would probably result in a more realistic overall consumption of the city.

7.3 Summary

This work investigated many different aspects regarding Cities:Skylines. At first, the RealtTimeMod (section 2.3) and its impact on the realism of the simulation were explained. Furthermore, this modification was sped up from 2.8 minutes per simulated hour up to 36 seconds per simulated hour during this work. In the following chapter ,3.1, the techniques used to create modifications were introduced and explained. The UML graph in figure 3.1, therefore, shows the manager design pattern used in Cities:Skylines. Based on this, the principle and development of the two mods created for this work were explained and proven. The CSExtractDataMod can extract data from within the simulation, such as electricity consumption, water consumption, heating consumption, sewage emission, and garbage emission. On the other hand, the CSImportDataMod can import the same parameters into the game using C# reflection techniques. Especially the CSImportDataMod implements a simple proof-of-concept algorithm (with its pseudocode shown in algorithm 1) for adapting the simulation.

In the second part of this thesis, three experiments were conducted. The first one showed the overall performance of the simulation environment Cities:Skylines with installed mods, but without injecting any real-world or random data into it. As a result, the simulation passed some of the predefined sanity checks regarding electricity, water, sewage, heating, and garbage consumption. However, the traffic density and traffic noise emission did not pass a single check and should be enhanced for further use. During the second experiment, the CSImportDataMod was tested. Therefore, a set of random numbers were imported as the momentary electricity consumption. After the simulation was fitted to the random values, the simulation was recorded. Finally, the recorded values were compared with the random data that was imported before. Its result demonstrated the import of almost any matter into Cities:Skylines. However, it also revealed outliers due to the adaption, as some districts had an enormous electricity consumption at single hours. These outliers can be seen in figure 5.3. Finally, the last experiment tested Cities:Skylines adaptability to real-world data. Therefore, the kWh values for an average day of each district were thankfully provided by the TraveNetz GmbH. These data were injected into the simulation using an adapted CSImportDataMod. In contrast to the second experiment, the error values were significantly lower, and it did not show any extreme outliers. Therefore, the mean error value of the last experiment resulted in about 11%. This experiment showed that Cities:Skylines can be adapted using a relatively small data basis without losing its properties pointed out in experiment I.

To conclude this work in one last sentence: Although these experiments showed that it is far from perfect, Cities:Skylines provides a remarkable well basis for smart city simulation in many respects with a focus on data privacy and should be considered for further investigations.

Bibliography

[1] Abosaq, N. H. Impact of privacy issues on smart city services in a model smart city. In: *International Journal of Advanced Computer Science and Applications* 10(2):177–185, 2019.

[2] Al-Rubaie, M. and Chang, J. Privacy-Preserving Machine Learning: Threats and Solutions. In: *IEEE Security & Privacy* 17(02):49–58, 2019. ISSN: 1558-4046. DOI: 10.1109/MSEC.2018.2888775.

[3] Burger, D., Baer, S., Dabelstein, L., Dohrendorf, M., Krumbiegel, L., Langentepe, M., Sternfeld, F., and Zirpins, B. Allgemeine Angaben und Naturverhältnisse - Lübeck in Zahlen 2018. In: Mar. 2019.

[4] *Cities:Skylines.* Accessed: 2021-09-14. URL: https://www.citiesskylines.com/.

[5] *Cities:Skylines Modding-API.* Accessed: 2021-08-30. URL: https://skylines.paradoxwikis.com/Modding_API.

[6] Duncan, C., Cunningham, J., Wang, A., and Kennedy, A. Urban Planning Optimization via "Cities: Skylines". In: 2021.

[7] Dwork, C. and Roth, A. The Algorithmic Foundations of Differential Privacy. In: *Foundations and Trends® in Theoretical Computer Science* 9(3–4):211–407, 2014. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: http://dx.doi.org/10.1561/0400000042.

[8] Fomby, T. Scoring measures for prediction problems. In: *Department of Economics, Southern Methodist University, Dallas, TX*, 2008.

[9] *Harmony.* Accessed: 2021-09-14. URL: https://harmony.pardeike.net/articles/intro.html.

[10] Hyndman, R. J. and Koehler, A. B. Another look at measures of forecast accuracy. In: *International Journal of Forecasting* 22(4):679–688, 2006. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0169207006000239.

[11] Karnouskos, S. and Holanda, T. Simulation of a Smart Grid City with Software Agents. In: *Computer Modeling and Simulation, UKSIM European Symposium on.* Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2009, pp. 424–429. DOI: 10.1109/EMS.2009.53. URL: https://doi.ieeecomputersociety.org/10.1109/EMS.2009.53.

[12] *Manager Design Pattern.* Accessed: 2021-09-14. URL: https://www.eventhelix.com/design-patterns/manager/.

[13] *Numpy.* Accessed: 2021-09-14. URL: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html?highlight=percentile#numpy.percentile.

[14] Olszewski, R., Cegiełka, M., Szczepankowska, U., and Wesołowski, J. Developing a serious game that supports the resolution of social and ecological problems in the toolset

environment of cities: Skylines. In: *ISPRS International Journal of Geo-Information* 9(2):118, 2020.

[15] Pinos, J., Vozenilek, V., and Pavlis, O. Automatic Geodata Processing Methods for Real-World City Visualizations in Cities: Skylines. In: *ISPRS International Journal of Geo-Information* 9(1), 2020. ISSN: 2220-9964. DOI: 10.3390/ijgi9010017. URL: https://www.mdpi.com/2220-9964/9/1/17.

[16] *RealTimeMod*. Accessed: 2021-09-14. URL: https://github.com/dymanoid/RealTime.

[17] *Reflection in .NET*. Accessed: 2021-04-10. URL: https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection.

[18] Richter, K. Precision of city simulations using publicly available city data. In: 2021.

[19] Suciu, G., Butca, C., Dobre, C., and Popescu, C. Smart City Mobility Simulation and Monitoring Platform. In: *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 685–689. DOI: 10.1109/CSCS.2017.105. URL: https://doi.ieeecomputersociety.org/10.1109/CSCS.2017.105.

[20] Yu, F., Zhu, Z., and Fan, Z. Study on the feasibility of LoRaWAN for smart city applications. In: *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 334–340. DOI: 10.1109/WiMOB.2017.8115748. URL: https://doi.ieeecomputersociety.org/10.1109/WiMOB.2017.8115748.