A Digital Twin for Ground Water Table Monitoring

LUIS RODRIGOANDRÉS MORALES ORTEGA
August, 2023

SUPERVISORS:
Dr. M. N. Koeva
Dr. ir. L. L. olde Scholtenhuis

A Digital Twin for Ground Water Table Monitoring

LUIS RODRIGOANDRÉS MORALES ORTEGA
Enschede, The Netherlands, August, 2023

Specialisation: Urban Planning Management

ABSTRACT

To cope with climate challenges, city engineers are redesigning spaces and infrastructures worldwide. In particular, two extreme situations confront municipal engineers and water boards: on one hand, in periods of rainfall shortage, an extremely low water table creates issues as vegetation roots cannot be supplied sufficiently with water. On the other hand, during heavy rainfall, high Ground Water Table GWT can emerge, leading to water entering into old buildings (an effect known as ingress). Traditionally, municipalities install pumps that run with a constant capacity to pump groundwater out to solve this issue. However, when extreme drought and rainfall events occur, the need to tailor monitor and proactive control of GWT throughout groundwater pump capacity is required to manage the anticipated rainfall or drought scenarios (Iris, 2022).

Examples in the academic literature illustrate that the Digital Twin (DT) concepts can help control the heating, ventilation, and air conditioning of buildings (Chen et al. 2022), or plan and forecast maintenance activities on bridges (Gao et al. 2023). Therefore, the application of this solution to GWT monitoring and control generated interest. However, DTs are full of challenges that go from collecting, management and storage of data, to accurate visualisation, monitoring and interaction for a desired purpose (Lehtola et al. 2022). Furthermore, completeness and resolution of data are required to make accurate simulations and predictions. Therefore, DT concepts remain prototypical and still need to progress to practice to date.

Therefore, this study explores digital twins applications in dynamic control of the GWT to address climate challenges such as excessive rainfall and drought. It aims to develop and evaluate a framework that integrates real-time data, prediction models, and connectivity with pumping machines to support dynamic GWT control. The process involves a 3D geospatial model that renders a city environment, incorporating GWT data from 277 sensors available from the public Twents Waternet website[1]**.** With it, a game engine uses a dynamic mesh to represent the GWT geometrically in the 3D model. An algorithm then predicts groundwater level based on rainfall data and uses the GWT level with a combination of Internet of Things (IoT) protocols as triggers in the 3D game engine to de/activate a water pump in the physical world.

The model was developed iteratively, and its usability was evaluated by urban drainage experts through a workshop that included a demonstration, a multidisciplinary discussion and a questionnaire (Annex M). The framework was tested in the city of Enschede in The Netherlands, and the results showed that the digital twin can effectively visualise historic data, current and anticipated changes in the water table and facilitate the identification of areas requiring pumping. The integration of forecasts allowed the prediction of possible GWT depths, visualise the possible scenario and support the decision process of pumping or retaining the GWT through messages or physical actions.

This study provided a proof-of-concept of the possibilities of the application of game engines and Digital Twins for GWT monitoring and control. It is recommended to test and improve the proposed solution by (1) visualisation principles and logical methodologies, (2) developing intuitive user interfaces, (3) identifying the strategic positioning and coverage of sensors and pumps across the urban area and (4) refining the predictive functionalities to enhance decision-making allowing a proactive preparation in anticipation of forthcoming meteorological events.

Keywords: Digital Twins, groundwater table, 3D modelling, real-time data, visualisation, decision-making, machine learning, prediction, interaction, game engine, climate challenges, IoT, Unity, automation.

[1] Twents waternet. 2022. "Home - Twente Water Network." Retrieved January 18, 2022 (https://www.twentswaternet.nl/).

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my family and friends for their unwavering support throughout this journey. Even during the most challenging moments, their encouragement and assistance helped me navigate the obstacles beyond the realm of research.

My heartfelt appreciation goes to Dr. Mila Koeva and Dr. Léon olde Scholtenhuis for their invaluable supervision and guidance throughout the course of this research. Their insightful input and steady direction consistently steered my work in the right direction, shaping the research into its final form. It was an absolute pleasure collaborating with them, and I am profoundly grateful for their mentorship and expertise.

I extend my thanks to my colleagues at the Municipality of Enschede for their indispensable contributions to this study. Their willingness to share data, offer insights, and provide ideas significantly enriched the research. Special recognition goes to Lieneke Stam, Marc de Jong, and Petra Klein for their outstanding support and collaboration.

Additionally, I wish to express my gratitude to Ing. Robert Godo Levensen and Lic. Eugenia Barillas for their instrumental role in encouraging me to pursue my master's studies. Their belief in my potential and their encouragement played a pivotal role in this journey. My sincere appreciation also goes to Prof. Karin Pfeffer and Prof. Dessislava Petrova-Antonova for generously agreeing to be part of the Assessment Board. Their insightful feedback did and will enhance the quality of this research.

I am grateful to various organisations that played a pivotal role in the completion of this research. The support of the Government of The Netherlands, the Orange Knowledge Programme from Nuffic, the Gemeente Enschede, University of Twente/ITC, Twents Waternet, KNMI, OpenAI, the Unity community, QGIS community, and ESRI was invaluable and deeply appreciated.

Finally, I wish to acknowledge the countless individuals, mentors, and resources that have contributed to my academic and personal growth. Their influence and assistance have shaped my perspective and enriched my journey.

In conclusion, this thesis would not have been possible without the collective support, guidance, and encouragement of these individuals and organisations. I am truly grateful for their contributions.

In Memoriam:

During these unprecedented times, we mourn the loss of dear friends and family members who left us during the pandemic. Their absence is deeply felt, and their memory will forever remain in our hearts. The challenges we faced collectively have reminded us of the fragility of life and the importance of cherishing the moments we share with those we hold dear.

Though their physical presence may be gone, the impact they made on our lives and the memories we shared with them continue to inspire and guide us. As we navigate these trying times, we honour their memory by embracing the lessons they taught us and carrying forward the love, strength, and resilience they exemplified.

This work is dedicated to their memory and the indelible mark they left on our lives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Background

According to Sustainable Development Goal 11 of the United Nations, Sustainable cities and human settlements, promoting sustainable land-use planning/management as well as integrated provision of environmental infrastructure have been established as a high priority. Under these categories water, sanitation, drainage, and solid waste management are included (United Nations 2020). Additionally, contemporary climate problems have led to the redesign of spaces and infrastructures in cities around the world. In The Netherlands, blue and green solutions such as water reservoirs and trees, are being re-integrated into public space at the expense of heat and water-retaining pavements. Besides these challenges, the cities expect to have an increasing problem coping with groundwater table issues (Normeninstituut Bomen 2022).

1.1.1. Groundwater Table

The groundwater table (GWT), is identified as the dynamic level below the surface at which the soil porosity allows the pass of water keeping it saturated. GWT should not to be confused with the term groundwater level (GWL) that is mostly applied to the sea level or the static water level of wells, as the latter can be higher than the GWT (Simple Pump, 2020). In particular, two extreme situations confront municipal engineers and water boards: on one hand, in periods of rainfall shortage, an extremely low water table creates issues as vegetation roots cannot be supplied sufficiently with water. On the other hand, during heavy rainfall, high GWT can emerge leading to water ingress in cellars under old buildings. To handle these situations, the municipalities need innovative solutions to tailor the existing groundwater pump capacity to the anticipated situations of rainfall and drought (Iris, 2022).

Therefore, the importance of monitoring GWT comes from the fluctuations that can happen due to drought periods or short, severe precipitations. For instance, in areas with highly built-up or paved environments, prolonged precipitation may saturate the ground over several days causing rainwater to form puddles on the land and contributing to waterlogging in urban areas generating wet basements and rising dampness (Klimaateffectatlas, 2022). Additionally, as the deepness of the tree's roots is determined in part by the distance between the surface and the GWT, prolonged exposure to droughts can force the tree's roots to extend until they reach a saturated level once more (Iris, 2022).

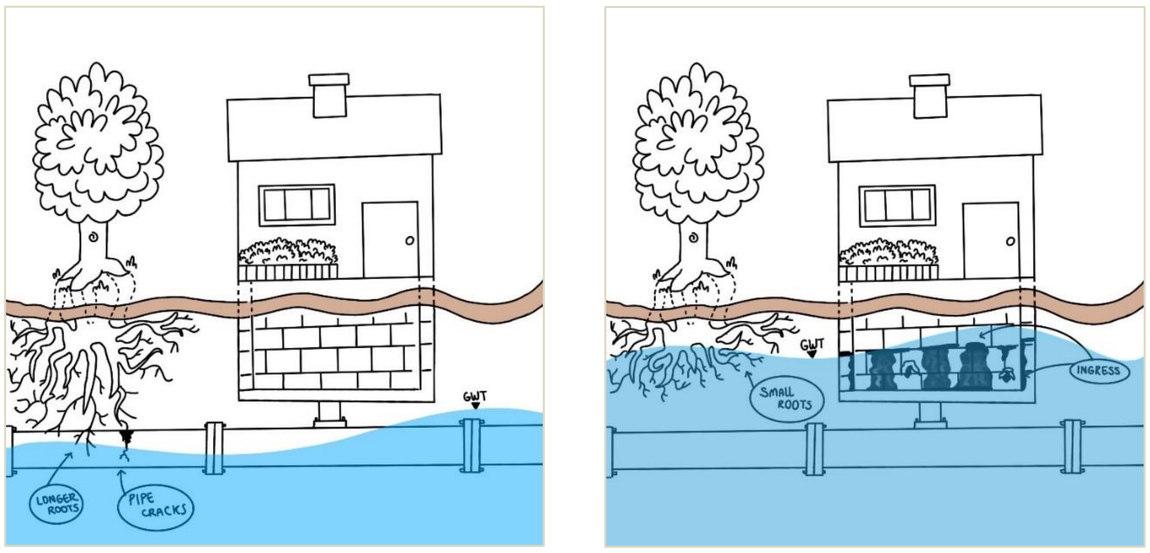


Figure 1. 2D profile to visualise the drought and ingress scenarios.

1.1.2. Groundwater Table Management

Groundwater management is a major challenge for cities, especially cities in the delta area. The fluctuations in the GWT are due to climatic events such as droughts and heavy rains, which have become more frequent and intense as a result of climate change. These fluctuations pose various problems for urban areas, including waterlogging and building flooding (Klimaateffectatlas 2022).

One conventional approach to groundwater table (GWT) management involves the installation of groundwater control pumps (Choo et al. 2021). However, this method often relies on fixed pump capacities, which may not be suitable for extreme situations. In periods of low rainfall, the GWT can drop significantly, resulting in inadequate water supply to vegetation roots and potential obstruction of underground structures such as pipes and cables. This can lead to plant stress and saturation of roots, making it challenging to carry out maintenance routines effectively. Conversely, during heavy rainfall, a high GWT can negatively impact tree root depth, resulting in shallow roots that are susceptible to wind-induced toppling. Additionally, high GWT levels can lead to water ingress in basements, causing damage to foundations and increased vulnerability to flooding (Normeninstituut Bomen 2022).

To address these challenges, cities around the world are exploring a dynamic and flexible approach to GWT management. For example, in response to increasing annual torrential rains and river flooding in Korea, the government is expanding a new drainage pump station to mitigate flood damage. However, adjusting the river level for the operation of the pump station has been a challenge due to the torrential rains. To address this, a study developed a method to control the river level and tested its effectiveness using a stormwater management model (SWMM) in Busan, Korea. The model was calibrated using rainfall data from 2011 to 2021, focusing on ten major floods in Busan. The simulations revealed a significant decrease in flooding, with the Suyeong River experiencing an average decrease of 3018.2 m3 and the Oncheon River requiring supplementary measures due to structural issues. By addressing the structural problems, the average flooding in the Oncheon River was reduced by 194.5 m3. This cost-effective and efficient method offers a solution for reducing urban stream flooding in areas prone to severe damage caused by climate change (Choo et al. 2021).

1.1.3. Recent Advancements in Digital Twin Techniques

According to (Digital Twin Geohub 2022) Digital Twins (DT) have been seen as a digital replicas of the physical living environments that supports decision-making through the seamless integration of a variety of data and analytics techniques. As such, DT are not mere geometric (2D and 3D) representations of static assets but a dynamic/live model that represents their past, current, and future states.

One promising application for ground water control is by integrating real-time data, predictive capacity of models, and 3D visualisation. Such integration takes place in the Digital Twin literature. Digital Twins have found applications in various domains, demonstrating their potential for research and real-world implementation. Previous studies have explored their utilisation in areas such as early fire alarm systems (Zhou et al. 2022) and urban regeneration projects (Eicker et al. 2018) showcasing the benefits of integrating real-time data into the virtual representation of assets. The ability of Digital Twins to monitor, analyse, identify problems, and predict outcomes based on different scenarios holds immense promise.

Technological developments often focus on expanding elements of Digital Twins (visualisation, prediction, sensory capabilities) or to apply the concept on a specific case. For example, one of the most ambitious projects in The Netherlands related to Digital Twins is the Netherlands3D which is an upscaled initiative from the 3D Amsterdam Digital Twin model (Gemeente Amsterdam 2023). This aims to create a national base platform for Digital Twins based on Unity Gaming Technology based on 3D BAG and the 3D Basisvoorziening (Basic Geo dataset), supplemented with local 3D dataset. This initiative started with the collaboration between the City of Utrecht, the City of Rotterdam, the Province of Flevoland and the Province of Utrecht, allowing each participant to retain developments initially based on its own organisational needs. The Netherlands3D project includes underground structure locations but the current version doesn't show models related to the groundwater table.

The previous examples show the increasing significance of advanced 3D model combinations in practical scenarios. These models exhibited remarkable research potential due to their adeptness in visualizing subterranean structures and analysing fundamental services. Moreover, these examples also underscore the transformative power of integrating real-time data and monitoring functionalities. Such integration can substantially enrich diverse analyses by enabling instantaneous and well-informed decisions through the evaluation of various scenarios.

As seen in (Lehtola et al. 2022) the main components of the DT:

- Dynamic/live representation of assets with past, current, and future states.
- Integration of real-time data from sensors for continuous analysis.
- Simulation and prediction capabilities based on real-world data.
- Utilisation of IoT, AI, and data analytics for enhanced outcomes.
- Support for informed decision-making and personalised solutions.
- Seamless integration with other components for feedback and evaluation.
- Virtual platform for exploring and establishing solutions in various fields.

1.2. Research Problem:

For water-related projects, the concept of Digital Twins (DT) has been used for the predictive maintenance of hydraulic electro-mechanical devices (Wang et al., 2022), energy efficiency of services (Onile et al., 2021), and water level behaviour prediction in a controlled environment (Edington et al., 2023). However, no previous research combines real-time GWT data with digital twins for physical control of the GWT.

Moreover, on the topic of GWT monitoring, a ground penetrating radar approach has been previously implemented to identify temporal variations on the GWT level as discussed by (Essam et al., 2020) in this case, no real-time data is implemented on a digital model, therefore its functions are merely for measuring purposes. There is a research that discusses the use of digital twins to monitor and identify predictive maintenance of hydraulic electro-mechanical devices (Wang et al., 2022) however, the scope of the research doesn't reach any physical control over the monitored equipment, therefore no direct interaction with the physical world is explored. Similarly, the work on 'Exploring the usage of Digital Twins for energy services and intelligent recommendations' is discussed in (Onile et al. 2021) where the authors apply consumer behaviour real-time data to predict and recommend policies based on the identified trends. Finally, the closest research identified so far that involves the use of Digital Twins in conjunction with physical mathematical models and a physical twin, in order to study the behaviour to predict water level behaviour throughout time, this research was performed on a self-contained environment composed by a reservoir and two tanks and is not taking the GWT in consideration (Edington et al., 2023).

The lack of research linking real-time GWT data to digital twins for monitoring and decision making has created a gap in the application of digital twin technologies for GWT management. This research aims to fill this gap by developing a model that integrates real-time GWT data from a sensor data portal with a digital twin model of the urban core area. The goal is to create an interactive visualisation of the GWT and enable energy-efficient decision-making to maintain stable GWT levels.

The table below summarises the main literature for Digital Twin related researches:

Research	Focus Area	Approach	Key Findings	Water Related	GWT Related
(Choo et al. 2021)	Flood Management	Stormwater management model (SWMM) and 3D visualisation	Developed a method to control river level, reducing flooding in Busan, Korea. The approach proved effective in mitigating flood damage caused by climate change.	Yes	No
(Zhou et al. 2022)	Early Fire Alarm Systems	Digital Twin integration with real-time data	Demonstrated benefits of Digital Twins in early fire alarm systems, leveraging real-time data for asset monitoring and problem identification.	No	No
(Eicker et al. 2018)	Urban Regeneration Projects	Integration of real-time data into virtual representation	Utilised Digital Twins for urban regeneration projects, showcasing their potential for research and real-world implementation.	No	No
(Gemeente Amsterdam 2023)	National Base Platform for Digital Twins	Integration of 3D datasets with Unity Gaming Technology	Netherlands3D initiative to create a national platform for Digital Twins, currently includes underground structures but not groundwater table models.	No	No

(Wang et al. 2022)	Predictive Maintenance	Digital Twin for hydraulic electro-mechanical devices	Applied Digital Twins for predictive maintenance of hydraulic electro-mechanical devices, but no physical control over monitored equipment.	No	No
(Onile et al. 2021)	Energy Efficiency of Services	Digital Twins with real-time consumer behaviour data	Explored Digital Twins for energy services and intelligent recommendations based on real-time data and trends.	No	No
(Edington et al. 2023)	Water Level Behaviour Prediction	Digital Twins combined with physical mathematical models	Studied water level behaviour in a self-contained environment (reservoir and tanks), not considering the GWT.	Yes	No
(Essam et al. 2020)	GWT Monitoring	Ground-penetrating radar approach for temporal variations	Implemented ground-penetrating radar for GWT monitoring, but no real-time data integration or Digital Twin utilisation.	Yes	Yes

Table 1. Available Digital Twin research documents comparison

In this section, we explored the significance of addressing fluctuations in the groundwater table (GWT) within urban areas. As climate change and environmental shifts continued, managing GWT became increasingly crucial. These variations had notable impacts on vegetation, flooding vulnerabilities, and the integrity of infrastructure. To address these challenges, solutions such as real-time monitoring, predictive models, and 3D visualisation through Digital Twins were employed. These approaches provided dynamic insights for sustainable urban planning, effective flood control, and enhanced environmental management.

1.3. Proposed Digital Twin Solution:

The study proposes the development of a digital twin framework using the Unity game engine that incorporates real-time GWT data from sensors distributed across the city. The framework provides an interactive visualisation of the GWT and aids in decision making by identifying areas that require automated water pump drainage and sends signals through the internet to activate the water pumps based on the GWT value, maintaining the GWT value levels under the surface. The framework also includes features such as historical data visualisation based on previous sensor readings and a predictive machine learning model that uses decision trees to predict GWT depth based on historical GWT depth and recorded precipitation data. These capabilities enable proactive decision making by providing insights into the potential impact of weather conditions on the GWT depth for the upcoming day. The proposed solution aims to close the GWT monitoring and decision-making gap by leveraging digital twin technologies, real-time sensor data, interactive visualisation and machine learning techniques. Research is expected to contribute to the development of an integrated and effective groundwater management and control system and facilitate sustainable water management practices in urban areas.

1.4. Research Objectives and Questions

General objective:

This research aims to develop a Digital Twin framework for groundwater table monitoring.

The following sub-objectives are considered during the research:

Sub-objective 1: To investigate what methods exist to generate Digital Twin models.
Questions to be answered:

RQ1.1: What methods currently exist for 3D modelling of surface and subsurface?
RQ1.2: What methods exist for real-time data integration?
RQ1.3: What sensor data is needed for a Digital Twin creation?

Sub-objective 2: To develop a framework for DT data integration as a monitor system with real-time GWT sensor data.

RQ2.1: What are the requirements of the stakeholders to develop a DT system?
RQ2.2: What methods will be used to generate the DT model?
RQ2.3: What is the available data for the case study area?
RQ2.4: How the interaction between the overground, underground and water sensor data will be achieved?
RQ2.5: What will be the final interface with the integrated data to link and use the physical world and the digital model?

Sub-objective 3: To simulate scenarios for monitoring, management and prediction of underground water levels

RQ3.1: What actions should be taken at high or low GWT scenarios?
RQ3.2: What are the needed data sources to support underground water level management decisions?
RQ3.3: What physics models will be used to predict appropriate GWT dynamics?

Sub-objective 4: To evaluate the developed GWT monitoring system workflow.

RQ4.1: Does the monitoring system meets the requirements of the stakeholders?
RQ4.2: What are the strengths and limitations of the developed workflow according to the stakeholders?
RQ4.3: What can be added to the monitoring system to enhance its applicability?

2. LITERATURE REVIEW

Digital Twin modelling has gained significant prominence as a powerful tool for understanding and managing complex urban systems (Lehtola et al. 2022). As cities continue to expand and evolve, the need for accurate representations of both surface and subsurface domains becomes vital. This literature review embarks on a comprehensive exploration of the methodologies employed in the creation of Digital Twin models, focusing on their applications in urban contexts. The main objective is to delve into the diverse methods utilised for 3D modelling of urban surfaces and sub-surfaces, as well as the mechanisms employed for real-time data integration, which is a fundamental element in ensuring the fidelity of these virtual counterparts. Additionally, the review aims to identify the details surrounding the sensor data that is essential for the foundation of Digital Twin models, shedding light on the types and quantities of data that facilitate the creation of accurate and responsive digital replicas. Through a meticulous analysis of existing literature, this review aims to provide a cohesive understanding of the approaches taken to construct Digital Twin models, ultimately guiding the subsequent phases of the research.

2.1. 3D modelling methods and applications towards digital twinning

In (Chopine 2011) a summarised compilation of the origins of 3D modelling is presented and showing that the core bases of it were conceived way earlier than computing devices were even created. The fields of architecture and engineering have widely embraced the adoption of 3D modelling as an essential tool. It enables the utilisation of diverse methods to recreate, identify, classify, assess, analyse, validate, plan, and simulate solutions for project planning, prevention, and execution. In these disciplines, 3D modelling serves as a powerful tool for designing and visualizing structures, facilitating efficient project management, and enabling accurate simulations for various scenarios. Its versatile applications contribute to improved decision-making, enhanced project outcomes, and streamlined workflows in architecture and engineering. (FutureLearn, 2022).

2.1.1. Advanced Models

Currently, several methods are employed in the creation of Digital Twin models for the 3D modelling of both surface and subsurface domains within urban environments. These methods encompass a range of techniques, each tailored to capture different aspects of the complex urban landscape. Some Digital Twin models employ a hybrid approach, integrating data from various sources such as GIS databases, BIM models, and ground-based surveys. This combination of data types ensures a comprehensive representation of both surface and subsurface features (Xue et al. 2020).

These 3D models can be processed at different spatial scales as shown by (Xue et al., 2020) that range from the general (i.e. global, and national level) to the more detailed ones on a regional or individual level of element representations. Even though the usage of 3D models has been spreading and adopted by several interdisciplinary fields, it is still a topic with the potential to be explored and expanded. Advanced 3D models like Building Information Modelling (BIM) and Digital Twins (DT) have gained traction either as research topics or research tools in recent years (Bucchiarone 2022; Digital Twin Geohub 2022)

BIM offers intelligent geometrical objects that can dynamically update project changes, encouraging collaborative visualisation and comprehension. It facilitates seamless teamwork across multiple disciplines, from planning to execution, and enables simulations within a digital environment to analyse established processes. However, as stated by (Digital Twin Geohub, 2022) “DT is not a mere geometric (2D and 3D) representation of static assets but a dynamic/live model that represents their past, current, and future states.”

Digital Twins allow for a better understanding of a project's performance while integrating data in real-time, with the ability to operate multiple simulations in order to identify problems or areas of improvement and predicting outcomes based on different scenarios. It is important to note that the data used with DTs can be generated by sensors installed in key stages, locations or elements for the functionality of a project (IBM, n.d.).

2.1.2. Game Engines

The concept of a game engine was introduced by ID Software in 1993 with the release of DOOM, marking a significant development in the gaming industry. Game engines separate game resources from player accessibility, revolutionizing game design on different platforms. Game engines are extensible software that serves as a foundation for multiple games, allowing developers to focus on other aspects of game development (Gregory and Lemarchand 2015). They provide essential functions such as graphics, audio, real time physics calculations, and AI, playing a crucial role in video game creation and development.(Li et al. 2023)

Considering the concepts and applications done in (Clausen, Ma, and Jørgensen 2022) game engines are becoming increasingly important for digital twins due to their advanced abilities to create immersive and interactive virtual environments. Adding to the previous concepts, here are some reasons why game engines are used to create digital twins:

No.	Feature	Description
1	Realistic Visualisation	Game engines excel at rendering high-quality graphics, making complex digital twin models easier to visualise and understand. They accurately represent physical properties.
2	Interactive Simulations	Provide real-time interactivity, allowing users to interact with and manipulate digital twin models. This improves the user experience and enhances understanding.
3	Physical and Dynamic Simulation	Offer robust physics engines that accurately simulate real-world physics, enabling realistic simulations in a digital twin environment.
4	Cross-Platform Deployment	Can run on various platforms, making digital twins accessible on desktops, consoles, mobile devices, and VR headsets, facilitating collaboration and sharing.
5	Intuitive Development Tools	Provide intuitive development environments and tools, including visual scripting systems and scene editors, for efficient creation and customisation.
6	Integration of IoT and Sensor Data	Support integration with IoT devices and sensor data, allowing digital twins to utilise real-time data for accurate simulations of physical systems.

Table 2. Game engine characteristics that make them suitable for Digital Twins (Kite-Powell 2022)

Examples of game engines in combinations with digital replicas show their potential for research and application porpoises. The research paper presented by (Horton et al. 2019) demonstrates a visualisation framework designed for simulations and real-time data exploration in virtual reality (VR) environments supporting multiple programming languages and the integration with the popular materials simulation engine LAMMPS. The proposed framework utilises common game engines like Unity and Unreal Engine to simplify VR headset access and uses the programming capabilities of these game engines, such as C# in Unity and C++ in Unreal Engine, to enable the reuse of existing simulation codes. This allows for real-time visualisation of simulation results within the game engine environment.

Another interesting application of game engines for data visualisation has been shown by (Li et al. 2023) where the authors developed a mixed reality method for underground pipelines in digital mechanics experiments. The visualisation integrates BIM models in the design phase and an algorithm for sensor data analysis. Human-computer interaction methods were created using game engines and mixed reality toolkits. Therefore, the study demonstrates the data perception of the underground pipe network and its mapping to the real environment, offering insights for engineering digital experiments.

A recurring game engine in the presented researches is Unity's. Its characteristics, advantages and limitations were explored by (Program-Ace 2021) the following table summarises their findings:

	Unity	**Unreal Engine**
Advantages	• Streamlined development for mobile apps: Unity was designed with mobile app development in mind, offering optimised tools and workflows for creating both 2D and 3D mobile games. • Support for immersive apps and prototyping: Unity provides robust support for VR, MR, and XR technologies, making it suitable for creating immersive experiences. It also offers a flexible environment for prototyping and experimenting with various functions. • Extensive asset store: The Unity Asset Store boasts a vast collection of ready to use assets, including 3D models, animations, scripts, and plugins, which can significantly speed up development.	• Geared towards achieving the highest quality of graphics: It excels at delivering stunning graphics with advanced rendering capabilities and a huge library of pre-built assets. • Allows users to customise shaders without coding: Unreal Engine's Blueprint visual scripting system allows developers to create complex game logic without extensive programming knowledge, allowing for rapid iteration and prototyping. • Lightning-fast post-processing: Unreal Engine's post-processing effects ensure fast and efficient rendering, allowing real-time adjustments and improving visual quality.
Limitations	• Graphics may require additional tinkering: While Unity provides some impressive graphical features out of the box, achieving top-tier visuals may require additional customisation and optimisation. • Smaller asset variety compared to Unreal Engine: Although the Unity Asset Store is extensive, it may not offer the same breadth of high-quality assets as Unreal Engine's marketplace. • Business-oriented plan costs: Free Personal plan, but more advanced and business-oriented plans available via subscription.	• Licensing applies a royalty system, requiring developers to pay a percentage of revenue once the project is monetised, which may be less favourable for larger commercial projects. • While Unreal Engine is capable of creating small-scale games, its advanced features and higher system requirements may make it less suitable for simpler projects with limited resources.

Table 3: Limits and Benefits of Each Engine (Program-Ace 2021)

Within the research community one of the key questions is which game engine to select. The research of (Program-Ace 2021) concluded that the selection depends on the objectives, budget and the size of the research. Additionally they also proposed the following selection criteria factors:

Factors	**Unity**	**Unreal Engine**
Project Requirements	• Cross-platform compatibility • Mobile game development • Extensive 2D and 3D support • Support for PC, consoles, VR, AR, MR	• High-quality graphics • Advanced rendering capabilities • Support for PC, consoles, VR, AR, MR
Development Expertise	• C# language • Visual scripting system • Broad range of learning resources and community support	• C++ language • Blueprint visual scripting system • Active developer community and learning resources
Project Scale and Budget	• Free Personal plan • Subscription costs for advanced/business plans • Royalty system for monetised apps	• Free access to the engine • Royalty system for monetised apps
Community Support	• Massive asset store • Abundant learning materials • Strong community support	• Smaller marketplace, but still offers resources • Countless manuals and forums
Business Model	• Consider licensing costs and revenue-sharing models • Evaluate compatibility with project's monetisation strategy	• Evaluate licensing costs and revenue-sharing models • Assess compatibility with project's monetisation strategy

Table 4: Factors for Selecting the Proper Engine for a Project (Program-Ace 2021)

2.2. Real-time Data Integration

According to (Javaid, Haleem, and Suman 2023) a Digital Twin is a software representation of a physical object, capturing its unique identity and providing a digital counterpart of tangible items like engines, structures, solar farms, or even entire cities. It simulates various processes and predicts their performance based on real-world data. The Digital Twin can utilise technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), and data analytics to enhance outcomes. With comprehensive data, the Digital Twin can suggest personalised solutions in real-time considering a specific historical reference and statistics.

(Bucchiarone 2022) explains that the ideal DT concept encompasses real-time datasets, machine learning, reasoning, and simulation to support informed decision-making. By utilizing real-world experiences, DT creates a virtual world that actively engages students in activities such as production and learning. It comprises three key components: the physical product, virtual or digital attributes, and the connection between them. DT technology offers solutions to various challenges related to learning experiences and knowledge development

Therefore, as DTs have extensive applications across diverse fields, effectively enhancing communication and improving operational efficiency. These systems are ideal to seamlessly integrate with other components, enabling feedback and evaluation during the product development phase. They can serve as prototypes to assess the potential outcomes and behaviours of physical models. As a result, DTs offer a virtual platform to explore and establish solutions for various technological applications (Bucchiarone 2022).

The integration of real-time data within digital twin models has emerged as a critical aspect to ensure the accuracy and dynamic functionality of these virtual replicas. The following methods have been explored to facilitate the seamless combination of real-time data into digital twin frameworks.

(Charest and Rogers 2020)Suggests the utilisation of Application Programming Interfaces (APIs) which serve as intermediaries for data exchange between disparate systems. APIs provide standardised protocols and tools that facilitate seamless communication between software components. In this context, Python, a versatile programming language, emerges as a valuable tool for interacting with APIs. Developers often employ Python scripts to orchestrate API requests, retrieve real-time data, and subsequently integrate it into the digital twin simulation. Additionally, cloud computing solutions have garnered substantial attention for real-time data integration. Digital twins can plug into cloud-based platforms through direct coding, allowing them to dynamically access, process, and analyse real-time data. This enables the digital twin to receive live inputs from dispersed sensors and devices, enhancing its responsiveness and accuracy (Javaid et al. 2023).

As stated in (Javaid et al. 2023) the rise of the Internet of Things (IoT) has significantly contributed to real-time data integration strategies. IoT devices, interconnected through networks, can be directly integrated into the digital twin's code, enabling the capture and transmission of real-time data from various sources. This influx of data continuously enriches the digital twin's dataset, encouraging a faithful representation of the physical system.

Moreover, Machine Learning (ML) algorithms and advanced data analytics techniques, integrated via direct coding, play a pivotal role in processing and deciphering the inflow of real-time data. By incorporating machine learning models directly into the digital twin's architecture, it can discern patterns, learn from incoming data, and make informed predictions. Through this continuous learning process, the digital twin evolves its understanding of the physical environment, enabling it to offer valuable insights for improved decision-making (Javaid et al. 2023). For example (Pham et al. 2022) discusses the use of seven different machine learning models, where random tree (RT) and random forest (RF) outperformed other ML, such as decision stump, M5P, support vector machine (SVM), locally weighted linear regression (LWLR), and reduce error pruning tree (REP Tree) for groundwater level prediction in a drought-prone area.

Additionally, the paper (Zhang 2023) explores the development of regression models, including decision tree regressor and random forest regressor, for sales volume prediction in a take-out business. Using shared data, the study examines their accuracy and finds that random forest regression has the greatest accuracy. It also emphasises that while linear regression improves the logical aspect of forecasting, it cannot provide enough accuracy on its own. The author advises integrating it with other approaches for increased accuracy, objectivity for sales volume forecasting.

2.3. Sensor Data Automation and Integration with Web Applications and Services:

(Mi et al. 2017) indicated in their empirical study of IFTTT, automation and integration using web applications have become increasingly popular as these platforms provide a way for end users to connect and control various web services and IoT devices through conditional rules. They play a key role in streamlining processes, increasing efficiency and improving the user experience. Automation refers to the ability to perform tasks or activities automatically and without manual intervention, while integration involves connecting and merging different systems, applications, or services to work together seamlessly.

The advent of the Internet of Things (IoT) has further expanded the scope of automation and integration, enabling the connection of physical devices and online services. Equipped with sensors and connectivity capabilities, IoT devices can communicate with web applications and services, enabling automated actions based on predefined conditions. Automation and integration with internet applications and services, including the IoT, offers many benefits. They enable synchronisation of data and workflows across multiple platforms, reducing manual effort and increasing productivity. For example, the integration allows data collected from IoT devices to be sent seamlessly to web applications for analysis and decision making (Mi et al. 2017).

According to (Moore 2020) Platforms like ifttt.com (If This Then That) have become powerful tools for automation and integration. IFTTT (If This Then That) is a trigger-action programming (TAP) platform that enables users to create conditional rules in the form of "if A then B." It allows users to automate actions and integrate various web applications, IoT devices, and services. Triggers and actions are provided by IoT vendors and web service providers, and users construct applets by selecting triggers and actions from different third-party partner services.

(Mi et al. 2017) also gives an insight into different works and research related to IoTs. The concept of Trigger-action Programming (TAP) has gained significant attention within the realm of automation, particularly in smart home, smart building, and IoT/context-aware systems. Over the years, numerous studies have delved into the intricacies of TAP, investigating its applications and efficacy(De Russis and Corno 2015). Among these, the IFTTT (If This Then That) platform emerges as a prominent player, offering a commercially successful implementation of TAP (Huang and Cakmak 2015) .

(Nandi and Ernst 2016)undertook an insightful exploration of the human factor inherent in trigger-action programming within the context of smart homes. This study involved the analysis of a vast collection of IFTTT applets, totalling 67,000, to unveil the expansive potential for creating a myriad of trigger-action combinations. While IFTTT-like platforms, including Atooma, WigWag, Android Tasker, Zipato, Stringify, and WayLay, have sought to replicate the TAP paradigm, their adoption and popularity have often paled in comparison to the widespread usage of IFTTT. This dominance further underscores the prominence of IFTTT as the exemplar of commercial TAP platforms (Huang and Cakmak 2015).

Nonetheless, the task of efficiently acquiring, analysing, and disseminating sensor data across a wide spectrum of heterogeneous sources remains a formidable challenge. This challenge is magnified by the lack of a standardised architectural framework capable of aptly encapsulating the dynamic attributes of sensor data, thereby impeding its seamless accessibility for collaborative actions. In response, (Gadea, Ionescu, and Ionescu 2010) proposed a forward-looking web-based solution, supported by a publisher/subscriber architecture, that extends real-time access to sensor data to enable multi-user web-based collaboration. This proposed solution seeks to transcend the existing barriers by filling the architecture with the potency to seamlessly distribute real-time sensor data within dynamic collaborative environments.

2.4. Literature Review Summary

This section explores the significance of Digital Twin modelling in understanding and managing complex urban systems. The review delves into methodologies used to create these models and their applications in urban contexts. It emphasises the importance of 3D modelling in architecture and engineering and the advanced capabilities offered by techniques like Building Information Modelling (BIM) and Digital Twins. Game engines are highlighted as versatile tools for Digital Twins due to their capabilities in realistic visualisation, interactive simulations, physical and dynamic simulations, cross-platform deployment, intuitive development tools, and integration of IoT and sensor data. The integration of real-time data is crucial for accurate Digital Twins, and methods like APIs, cloud computing, Internet of Things (IoT) devices, machine learning algorithms, and data analytics are explored for this purpose. Automation and integration through web applications and services, particularly platforms like IFTTT, are discussed for streamlining processes and increasing efficiency. The challenges and potential solutions for acquiring and disseminating sensor data are also highlighted, along with proposed web-based architectures for collaborative access to real-time sensor data.

3. CONCEPTUAL FRAMEWORK OF PROPOSED GWT DT

By embracing this innovative approach, decision-making processes can be streamlined and made more effective, leading to sustainable and water-resilient cities. With the challenge clearly defined, the research endeavours to bridge the gap between traditional monitoring methods and cutting-edge technology, it is then possible to propose a framework that is able to empower stakeholders to understand, predict, and manage Groundwater Table (GWT) fluctuations in real-time. This endeavour aims to harness the capabilities of digital twins and game engines to create a comprehensive and efficient solution for urban water management.

3.1. Specifics of the Conceptual Framework:

The model presented is a product of the researcher's original work and design, drawing inspiration from the case-specific requirements and insights collected from relevant literature. It is important to emphasise that this model is not derived solely from existing literature, but rather reflects the researcher's innovative thinking in combining the specific needs of the case with relevant concepts and approaches from the literature. The researcher's unique contribution lies in developing a comprehensive system that integrates real-time data from sensors, creates a Digital Twin for visualisation, and employs predictive modelling techniques, thereby offering a novel solution for groundwater table monitoring and urban water management.

Real-time data is collected through a network of sensors distributed across the city to measure the groundwater table (GWT) depth. This data is utilised to update the Digital Twin, which represents the city in a spatial 3D model. Subsequently, simulations and actions are executed within the Digital Twin to generate information that facilitates decision-making and establishes optimal working ranges or conditions that can influence the physical world.

For instance, GWT data obtained from sensors can be utilised to simulate and visualise the GWT, allowing for the identification of the desired depth range. If the GWT falls outside of this ideal range, appropriate decisions can be made for each scenario. For instance, if the GWT level is excessively high, water pumps located throughout the city can be activated to lower it. Conversely, if the GWT level is too low, the water pumps can be deactivated to allow for natural recharge.

Moreover, weather data, including rainfall forecasts, can be incorporated into the measurements. By utilizing a predictive model, anticipated GWT depths for the following day can be estimated, enabling proactive planning and pre-emptive solutions rather than reactive responses.

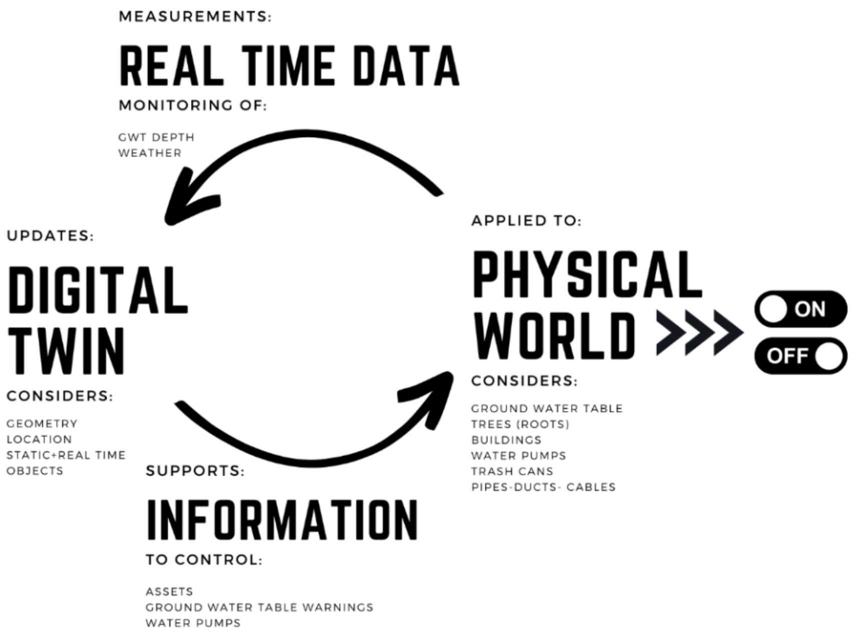


Figure 2. General conceptual framework for a GWT Digital Twin, explaining the flow of real-time data to achieve changes in the physical world

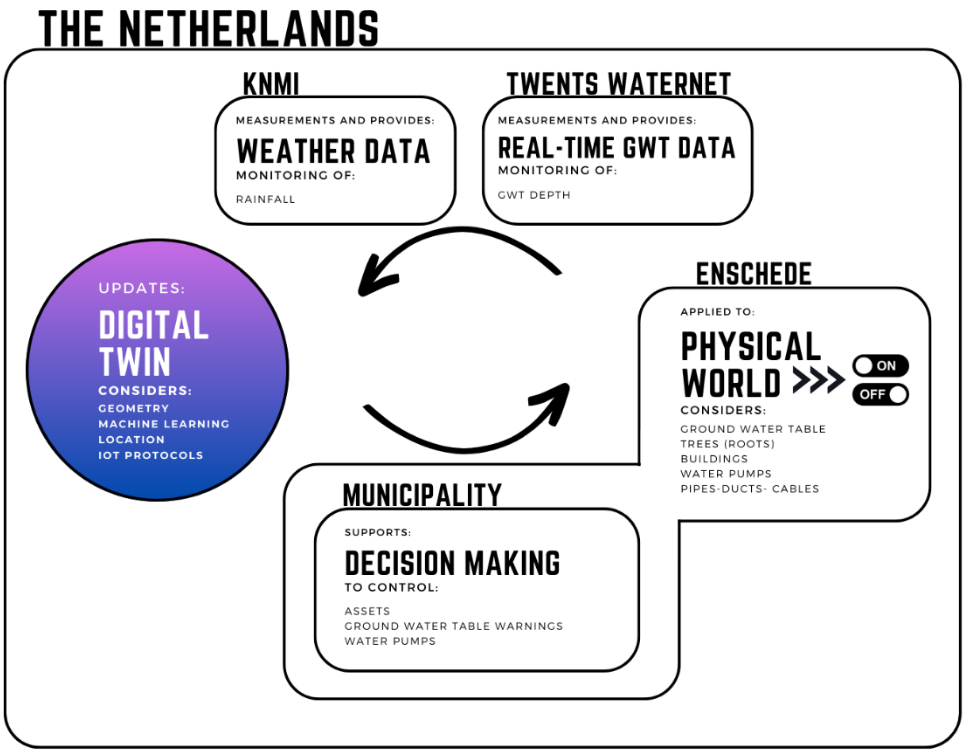


Figure 3. Conceptual framework applied to Enschede.

4. STUDY AREA

4.1. City of Enschede, The Netherlands

The City of Enschede, with more than 160.000 inhabitants, is one of the three largest municipalities located to the east of The Netherlands in the Overijssel province and Twente region. Enschede is surrounded by Dinkelland, Oldenzaal and Losser in the north, Hengelo in the east, Haaksbergen in the south and Germany to the east (Overijssel.nl n.d.).

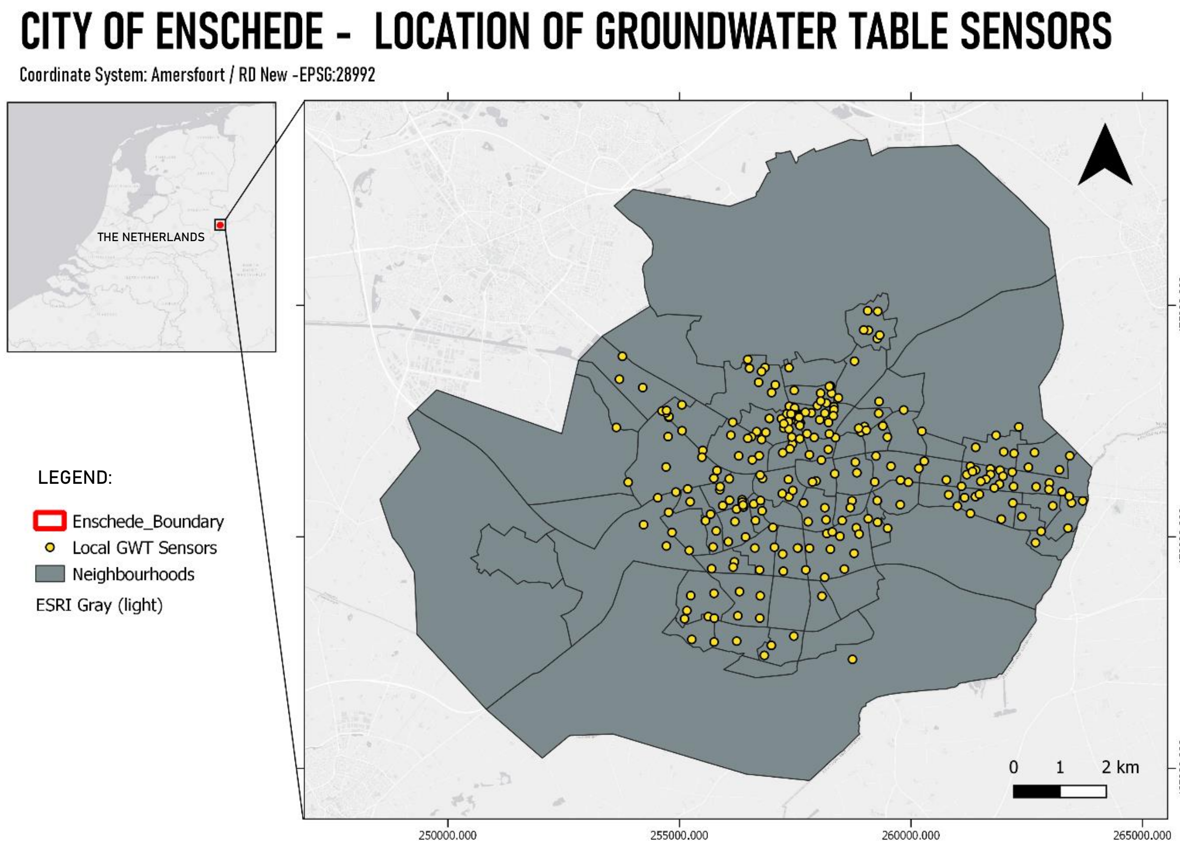


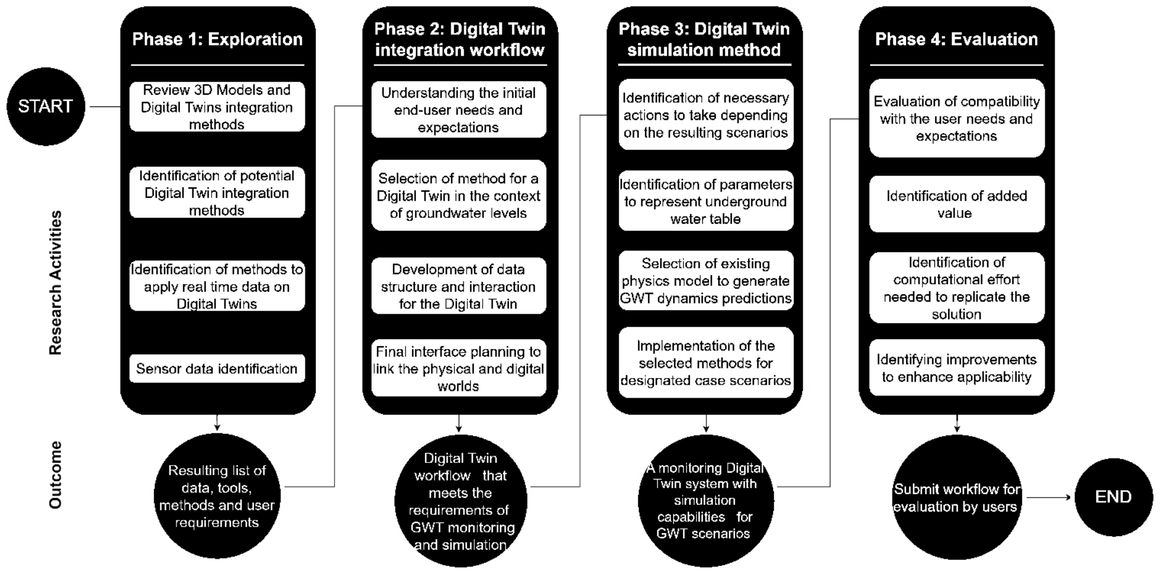
Figure 4. Map of the study area and sensor location in the city of Enschede in The Netherlands.

In the year 1860 the textile industry was Enschede's most important economic activity and thanks to a bloom in that market, the city saw a fast grow during that time. Due to the textile industry, water was pumped constantly lowering the groundwater levels, therefore drying up the source of streams.

Nowadays with the cease of the textile industry in the region, water is mayorly being pumped out of the ground by the Grolsch factory which targeted to use, in a conservative manner, 3.05 hl/hl (hectolitres of water per hectolitres of beer) to operate on the year 2020 (Slot 2019). This has helped to maintain the GWT levels in Enschede. However, both oversaturation and drought of land can prompt problems for the liveness of trees/vegetation, risk of waterlogging conditions and eventual floods in the region. Therefore, the Municipality of Enschede has adopted the responsibility to keep track of the groundwater table (GWT) levels and try solutions to keep it stable.

5. RESEARCH METHODS

The following methodological flowchart shows the different phases that are applied during the research, each phase is further explained to detail:



START
Phase 1: Exploration
Review 3D Models and Digital Twins integration methods
Identification of potential Digital Twin integration methods
Identification of methods to apply real time data on Digital Twins
Sensor data identification
Resulting list of data, tools, methods and user requirements
Phase 2: Digital Twin integration workflow
Understanding the initial end-user needs and expectations
Selection of method for a Digital Twin in the context of groundwater levels
Development of data structure and interaction for the Digital Twin
Final interface planning to link the physical and digital worlds
Digital Twin workflow that meets the requirements of GWT monitoring and simulation
Phase 3: Digital Twin simulation method
Identification of necessary actions to take depending on the resulting scenarios
Identification of parameters to represent underground water table
Selection of existing physics model to generate GWT dynamics predictions
Implementation of the selected methods for designated case scenarios
A monitoring Digital Twin system with simulation capabilities for GWT scenarios
Phase 4: Evaluation
Evaluation of compatibility with the user needs and expectations
Identification of added value
Identification of computational effort needed to replicate the solution
Identifying improvements to enhance applicability
Submit workflow for evaluation by users
END
Research Activities
Outcome

Figure 5. Methodological flowchart of the research process

5.1. Phase 1: Exploration

The methodology phase commenced with an extensive literature review to establish a foundational understanding of key concepts essential to the research. This included a comprehensive exploration of 3D modelling techniques (including game engines), Digital Twins, real-time data collection (including sensor data), groundwater table (GWT) dynamics, predictive models (including machine learning techniques) and different implementation methodologies. By conducting this initial investigation, profound definitions and keywords were identified, providing valuable insights that guided the subsequent stages of the research.

Furthermore, locating of pertinent data for the development of a Digital Twin involved an extensive search across both private and open data sources in the Netherlands, such as the Publieke Dienstverlening Op de Kaart (PDOK). The identified data was meticulously gathered, organised, and stored, taking into consideration factors such as data recoverability and proper classification based on permissions and licensing requirements. For a comprehensive understanding of the data management process, refer to **Annex A: Data management plan.**

Additionally, a preliminary examination of potential methods for simulating groundwater levels was conducted through an in-depth analysis of existing literature. This investigation aimed to establish a theoretical foundation that would inform initial decisions and provide guidance throughout the study.

5.2. Phase 2: Digital Twin integration workflow

In the subsequent phase of the research, a careful selection was made to determine the most appropriate method for the study. This decision-making process took into consideration potential stakeholders requirements and research objectives. It involved considering various factors such as the choice of 3D modelling software, real-time data integration, open-source tools, plugins, and methods for Digital Twin creation. By thoroughly evaluating these options, it was aimed to ensure that the selected approach would align with the specific needs of the research.

A comprehensive approach was undertaken to integrate a 3D model with real-time data in this research. Firstly, a method was developed to iteratively locate, download, organise, and store the latest .CSV files from available from the Twents Waternet website. These files contained height measurements of the groundwater table (GWT) obtained from various sensors in the study area, providing the most up-to-date values for the modelling process. A key aspect of the methodology was the ability to set the iteration of downloading the data at a desired repeating time interval. This flexibility allowed for customisation of the refresh rate in the Digital Twin according to specific requirements. For instance, the data could be retrieved every 5 seconds, 30 minutes, or 1 hour, among other options. The chosen interval depended on the desired refresh rate needed to achieve real-time representation in the Digital Twin.

It is important to note that the refresh rate was also influenced by the availability and posting frequency of the data on the Twents Waternet website. The research took into account the limitations imposed by the data source and its update frequency to ensure reliable and up-to-date information was integrated into the Digital Twin.

To create an interactive modelling environment, a game engine approach was adopted, with Unity being selected as the appropriate software. Unity offers advanced real-time rendering techniques, including Occlusion Culling, which optimises performance, especially for large-scale models like the city centre. Moreover, Unity's integrated coding capabilities through Visual Studio's C# enabled seamless interaction with real-time data and external communication to Internet of Thins (IoT) equipment. This flexibility and open-source nature of Unity made it a suitable tool to accomplish the research objectives effectively.

Furthermore, a 3D representation model of the City of Enschede was created using CityEngine, a software specialised in procedural modelling. To ensure compatibility with Unity, it was decided to export the model in the FBX format. This format preserves important features of the original model, including textures, spatial reference within Unity, and the ability to treat each building as a separate entity. And to ensure the practical applicability of the research, the interaction between real-time groundwater table data and the generated digital model was explored. This exploration allowed identification of potential interactions that could be simulated on a physical model, thus guiding the selection and adjustment of the Digital Twin workflow. Notably, the requirements for flood and drought simulations were considered to ensure the effectiveness of the proposed approach.

A Delaunay triangulated mesh was chosen as the representation of the GWT, utilizing the available sensor data. This mesh was calculated in QGIS and translated into Unity coordinates, where each sensor was represented by the vertices of the triangulated mesh. This approach allows for rendering an approximation of the groundwater level between measured points by adjusting the depth variable.

The used data contains the GWT height measured from the reference height in The Netherlands NAP and has a temporal resolution of 1 hour between validated measurements. This data is then combined with weather forecast data to generate a (very) basic predictive model that forecasts changes in the GWT. The collected and integrated data consisted of real-time information on groundwater conditions obtained from the public measurement portal Twents Waternet, weather forecast data from a website like Buienradar, and historic rainfall data (per hour) from the National Knowledge Institute for Weather, Climate, and Seismology (KNMI). This predictive model is open for the user to choose and substitute as needed, as the predictive model itself is not the main focus of this research, but for proof of concept purposes the use of a machine learning algorithm was selected to generate applicable testing values. The digital twin then is used to visualise current and anticipated changes in the water table and help identify the designated areas where pumping may be needed.

An integrated workflow was developed by combining various components based on the selections made during the process. However, to effectively implement a GWT Digital Twin, an additional factor related to the control of IoT equipment in the real world was necessary. In order to achieve this, three main elements needed to work together: actuators, data, and a visualisation and control interface. For the implementation case, a domestic water pump was chosen as a proof of concept. In addition, the IoT protocol control was managed through Unity, along with an API key for the website ifttt.com. This allowed the power supply of the water pump to be turned on or off using a smart plug.

Lastly, the visualisation and control tool utilised the Unity game engine, and the data was accessed through .CSV files via active running C# code. This combination of components and technologies enabled the successful implementation of the GWT Digital Twin.

The goal was to establish a cohesive and effective methodology for preparing 3D models with Digital Twin capabilities, specifically tailored to the context of groundwater levels.

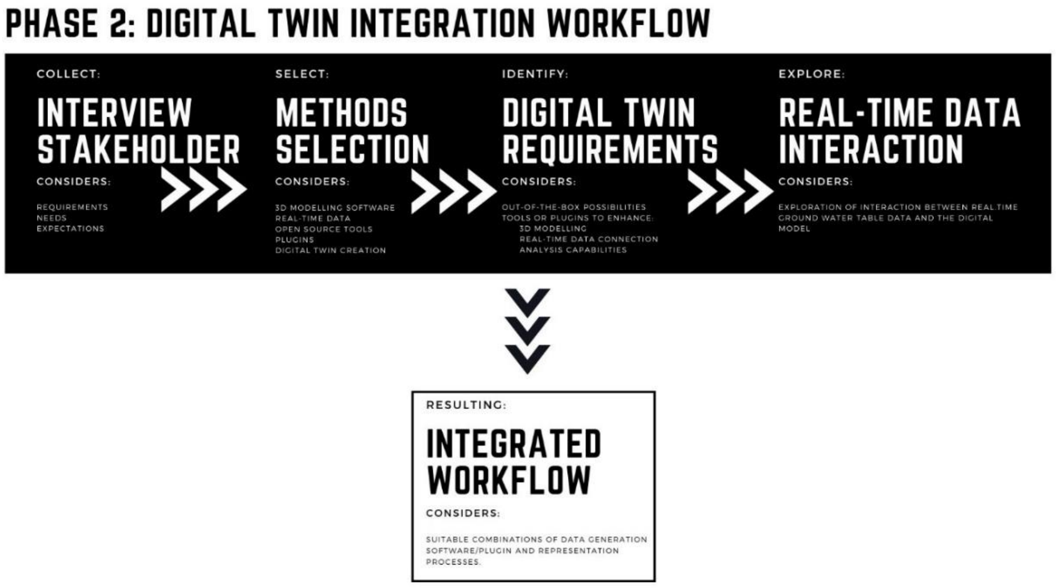

PHASE 2: DIGITAL TWIN INTEGRATION WORKFLOW
COLLECT:
INTERVIEW STAKEHOLDER
CONSIDERS:
REQUIREMENTS
NEEDS
EXPECTATIONS
SELECT:
METHODS SELECTION
CONSIDERS:
3D MODELLING SOFTWARE
REAL-TIME DATA
OPEN SOURCE TOOLS
PLUGINS
DIGITAL TWIN CREATION
IDENTIFY:
DIGITAL TWIN REQUIREMENTS
CONSIDERS:
OUT-OF-THE-BOX POSSIBILITIES
TOOLS OR PLUGINS TO ENHANCE:
3D MODELLING
REAL-TIME DATA CONNECTION
ANALYSIS CAPABILITIES
EXPLORE:
REAL-TIME DATA INTERACTION
CONSIDERS:
EXPLORATION OF INTERACTION BETWEEN REAL TIME GROUND WATER TABLE DATA AND THE DIGITAL MODEL
RESULTING:
INTEGRATED WORKFLOW
CONSIDERS:
SUITABLE COMBINATIONS OF DATA GENERATION SOFTWARE/PLUGIN AND REPRESENTATION PROCESSES.

Figure 6. Phase 2 overview schematic view.

5.3. Phase 3: Digital Twin Simulation method

Identifying future changes in the groundwater table (GWT) is crucial for determining appropriate actions to be taken in response to potential scenarios. This includes recognizing both reactive and preventive solutions for situations where the GWT is either too low or too high. To accurately represent underground water levels, a set of geometrically descriptive parameters was identified. These parameters were based on available data or, if necessary, dummy data to facilitate scenario creation.

A static model depicting the City of Enschede was incorporated in the Unity 3D environment, while a dynamic mesh was developed to iteratively update its geometric characteristics over a specified time period (as the data refresh rate in the Twents Waternet is per hour, periods between 15 and 30 minutes are effective for this purpose). This was achieved by retrieving GWT depths from the sensor data and loading them into the mesh's Y axis position. The mesh was generated based on a Delaunay triangulation, serving as a interpolated visual representation of the GWT. Each sensor corresponded to the vertices of the triangulated mesh (X and Z positions) and were accurately positioned within the spatial reference of the 3D model. For analytical purposes, GWT forecasts were incorporated into an optional visualisable environment using results from the predictive model. These forecasts facilitated the identification of specific values required for generating future prediction models in subsequent iterations of the digital twin.

After doing the literature review on ML methods, a supervised learning technique was selected for the GWT depth prediction. While the Random Forest (RF) and Random Tree (RT) methods offer improved predictive accuracy and reduced overfitting through ensemble techniques (scikit 2023a), the Decision Tree Regressor was selected due to its simplicity and ease of interpretability. The decision tree regression is a supervised learning algorithm that builds a model in the form of a tree structure to make predictions. It works by recursively splitting the data based on different feature values to create decision rules that predict the target variable. Additionally, the Decision Tree Regressor allowed to directly observe the decision-making process and visualise the model's logic, which is particularly valuable when transparency and understanding of the model's behaviour are essential. Despite its potential limitations on complex datasets, the Decision Tree Regressor's ability to capture complex relationships (scikit 2023b) and its straightforward representation align with the goal of maintaining a comprehensible model and therefore allow users to implement the predictive models that align the most with their study area needs.

The Python code for the predictive model was developed and configured to run in a Google Colab environment. Using pre-prepared training data, the model is trained with data specific to each sensor. By employing supervised learning techniques, the predictive model generates forecasts of groundwater table (GWT) depth values for each sensor location for the next day. In this case, the model is trained using historical data of groundwater depth and precipitation (RH) values. The input features for training are the 'Date_Num', 'Hour_Num', and 'RH(mm)', and the target variable is the 'Depth(m)'. Then, the decision tree regressor is trained on a portion of the data and evaluated using the testing set. After evaluating the model's performance, it is used to predict the groundwater depth for the specified date and RH values. Overall, the code combines data from different sources and uses a decision tree regressor model to make predictions based on historical data. The code used in this predictive model can be found Annex L: GWT Decision Tree Predictive Model Code (Google Colab).

Incorporating the collision and trigger capabilities offered by the implemented game engine, a proof-of-concept was constructed to facilitate interactions and initiate actions. Specifically, when the digital representation of the groundwater table (GWT) made contact with the surface model of the city within the simulation, a command was dispatched over the internet to establish communication with a domestic water pump. This IoT-based protocol allowed for the activation or deactivation of the water pump based on the current GWT levels. In practice, this action entailed activating a smart plug, connected to the internet, which in turn powered on the water pump. As a result, the water level in the container housing the pump experienced a reduction showing a direct influence in the real-world. Conversely, when the GWT ceased to be in contact with the surface in the simulation, a second signal was emitted and transmitted through the internet to the water pump, causing it to deactivate. This mechanism ensured synchronisation between the digital GWT and the physical water pump, mirroring the behaviour observed in the simulation.

Moreover, this study has identified potential applications for flood and drought simulations, presenting avenues for future exploration and development. The integration of Digital Twins with real-time data for groundwater table (GWT) control holds promise for enhancing water management strategies in urban areas, particularly those susceptible to climate-related challenges. By harnessing the power of technology and data, cities can proactively monitor, analyse, and respond to fluctuations in the GWT, thereby fostering more sustainable and effective water resource management practices.

To facilitate the analysis and evaluation of predictions, the implementation of an optional visualisable environment allowed for the loading of either test or historical weather data. The test data serves the purpose of identifying optimal ranges necessary for generating future prediction models in subsequent iterations of the digital twin. The current iteration of the research primarily focuses on visualizing and controlling the forecasted dynamics of the GWT, involving the estimation of thresholds indicating the proximity of the GWT to the surface. These thresholds serve as triggering values to activate system messages or actions, such as activating or deactivating physical or digital pumps. The chosen parameters for consideration and programming within the 3D modelling software include GWT levels, elevation data, and weather forecasts, which are used to calculate and assess estimates of future events.

Subsequently, the selected methods were implemented in a designated case scenario that varied based on the thresholds of the underground water table. This allowed for the activation of different messages or actions within the monitoring system. The scenarios encompassed instances where the GWT was sufficiently high to make contact with the surface, situations where the predicted GWT height had a high probability of contacting the surface, and a default scenario where the water level was insufficient to reach the surface. The latter scenario served primarily as a monitoring tool for real-time and historical data analysis. These scenarios were programmed within the game engine software, enabling the utilisation of GWT values in conjunction with surface elevation data and weather forecasts as inputs. Consequently, estimates for future events could be calculated and evaluated according to the specific requirements and expectations of the end-users.

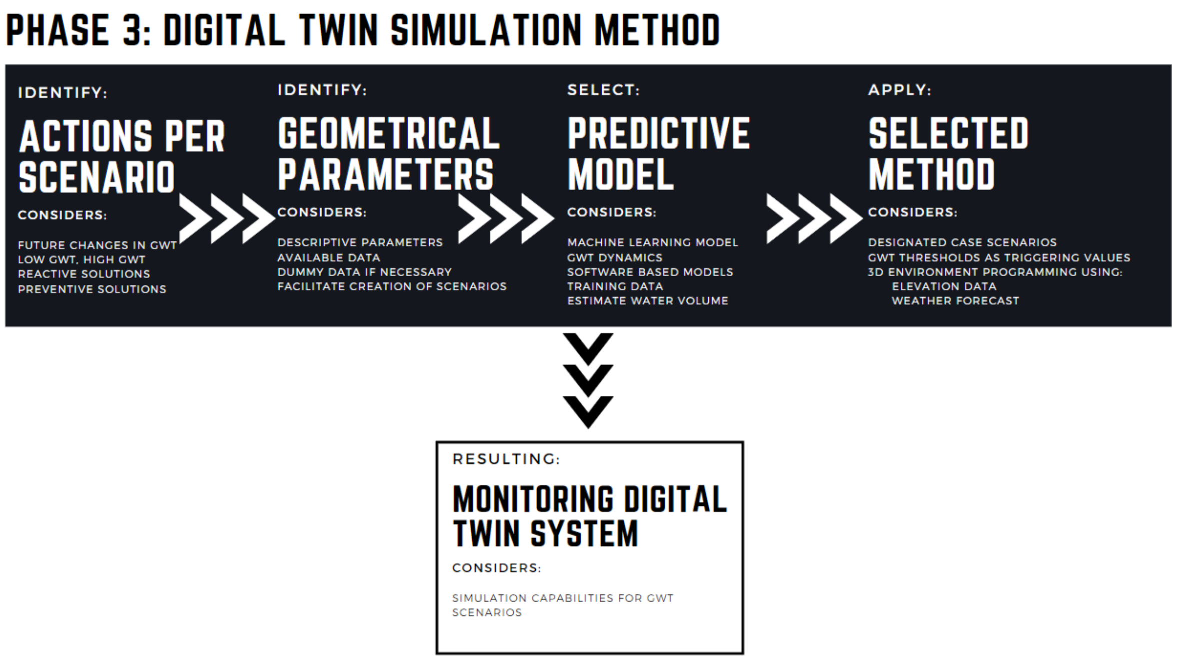


Figure 7. Phase 3 overview schematic view.

5.4. Phase 4: Evaluation

The initial version of the proposed methodology was subjected to evaluation from the University of Twente and the Municipality of Enschede. The evaluation aimed to gather data on the feasibility and usability of the model in different working environments, while also assessing its compatibility with the specific requirements and limitations of each institution.

The evaluation process focused on assessing the value added by the simulation method and considered several key factors, including the visual representation component, editing capabilities, integration of smart data, interoperability, and scalability.

To validate the accuracy of the GWT dynamics predictions, a comparison was made between the model-generated prognostications and the actual values obtained from the Twents Waternet sensors over a defined time period. This comparative analysis provided insights into the strengths and limitations of the developed method and highlighted areas for improvement, particularly regarding the establishment of physical connections with real in-situ water pump equipment and the user interface. The feedback obtained from this evaluation process played a crucial role in refining and adjusting the framework, with additional features being considered to enhance its applicability in future iterations.

Additionally, through testing the framework in lower specification equipment than the one it was developed, an evaluation was conducted to assess the computational effort required for implementing the monitoring system. This evaluation aimed to determine the minimum hardware requirements necessary to replicate the system while ensuring its reliability. Three distinct computers were utilised to evaluate the framework's performance (see Table 5). The MacBook White 2009, being notably outdated and less potent compared to the other two laptops, encountered difficulties in rendering the complete city model. The Lenovo L14 Gen4 served as one of the most recurring models utilised by students in the ITC during the research, showcasing a commendable equilibrium between performance and accessibility. Conversely, the AORUS 15P equipped with RTX 30 Series excelled in high-end gaming and rendering capabilities and was the primary system utilised for framework development. The assessment of "rendering capability" was a general estimation, with actual performance variability contingent on specific visualisation aspects within the framework. For prospective assessments, it may be worthwhile to consider testing with equipment possessing memory capacities of 8 and 16 GB RAM.

Specification	**MacBook White 2009**	**Lenovo L14 Gen4 (AMD)**	**AORUS 15P (RTX 30 Series)**
Operating System	Snow Leopard	Windows 11 Home	Windows 11 Home/Pro
Platform	Intel Core 2 Duo	AMD	Intel Core i7
Processor	Core 2 Duo 2.26GHz	Ryzen 5 PRO 7530U	Core i7 H-Series
Base Clock Speed	2.26 GHz	2.00 GHz	Varies
Display	13.3-inch (diagonal) LED	14" FHD IPS, 250nits	15.6" FHD 240Hz IPS
Memory	2GB DDR3 1066MHz	32GB DDR4-3200MHz	32GB RAM
Storage	256 GB	1TB SSD M.2 PCIe Gen4 TLC Opal	Varies
Network	Wi-Fi, Bluetooth	Ethernet port	-
Graphics	NVIDIA GeForce 9400M	Radeon RX Vega 7	NVIDIA GeForce RTX 30 Series
Rendering Capability	Basic	Moderate	High
Cores/Threads	2/2	4/8	Varies
Source	(Apple 2009)	(ITC 2021)	(Gigabyte 2021)

Table 5. Available equipment for framework test comparison.

5.5. Methods Summary

In the first phase of the research, a comprehensive exploration of various methodologies formed the foundation for subsequent stages. This encompassed an in-depth investigation of 3D modelling techniques, encompassing BIM, Digital Twins and GIS integration, revealing the unique dynamism of the DT. The integration of real-time data emerged as a fundamental requirement for Digital Twins, distinguishing them from traditional 3D models. Notably, Digital Twins were recognised as the dynamic approach enabling real-time interactive visualisation of phenomena like groundwater tables (GWT). Given this context, game engines stood out as suitable platforms due to their programmable nature, real-time data analysis capabilities, and compatibility with IoT protocols. Unity was chosen as it aligned with open-source principles, offering cost-effectiveness and platform adaptability. The second phase involved the integration of a 3D model and real-time data. A flexible .CSV file retrieval system was established, ensuring data refresh rates align with specific needs. Unity emerged as an appropriate choice due to its rendering techniques and coding capabilities. CityEngine was used for procedural modelling, enhancing compatibility with Unity. Subsequent phases unveiled the construction of a dynamic mesh representation of the GWT, enabled by sensor data integration and weather forecasts. Predictive modelling using a decision tree regressor compatible with Python was implemented to forecast GWT changes. The integration of IoT-enabled water pump control was achieved, enabling real-world interaction. Feedback from evaluation facilitated framework improvement, highlighting opportunities for flood and drought simulations and enhancing water resource management in urban areas prone to climate impacts. The proposed methodology demonstrated the potential of 3D models with Digital Twin capabilities to address real-world challenges, showcasing their value in environmental monitoring and decision-making processes.

6. RESULTS AND DISCUSSION

6.1. Investigation of methods for 3D modelling of surface and subsurface

In the first phase of the research multiple methods for 3D modelling of surface and subsurface were identified, encompassing both fundamental solutions such as CAD (Computer-Aided Design) and GIS (Geographic Information System), as well as more advanced approaches like BIM (Building Information Modelling) and Digital Twins. It was observed that the generation of Digital Twins (in comparison to BIM) necessitates the integration of real-time data with the model, and it is not constrained to a particular software platform. As stated by the (Digital Twin Geohub 2022), a Digital Twin is not merely a static representation of geometric assets in two or three dimensions, but rather a dynamic and live model that captures their historical, current, and future states. Consequently, the research journey led to the identification of Digital Twins as the dynamic template capable of facilitating an interactive real-time visualisation of the Groundwater Table (GWT). Building upon this foundation, it became evident that the incorporation of programable real-time data analysis and prediction, 3D modelling generation and visualisation, along with the capacity for triggering actions and communicating through IoT protocols, could be seamlessly coordinated within game engines. This realisation brought the focus to game engines as appropriate candidates for comprehensively constructing a Digital Twin framework.

Furthermore, aligning with the research's predominant aim of embracing openness and accessibility, the adoption of the Unity game engine emerged as the optimal choice. The rationale behind this selection stems from Unity's cost-effective licensing, which resonates with the open-source principles. Notably, Unity's compatibility across multiple platforms enhances the framework's versatility, rendering it adaptable to diverse environments and ready for potential future refinements.

From the literature it was concluded that currently, there is no one comprehensive out-of-the-box software solution available that offers seamless integration with real-time data. Although some intermediary solutions attempt to bridge this gap, it is still necessary to customise and establish connections between real-time data and the model based on each individual project requirements, available data sources, and formats. Thus, three essential components must be identified and interconnected for real-time data integration. Firstly, the collection and preparation of data are crucial, typically achieved through sensors that provide frequent updates on the relevant measurements of interest. Secondly, a model capable of presenting the collected data in an interactive manner. And finally, there needs to be a linkage established between the aforementioned components. This linkage can be accomplished through iterative coding techniques, which involve reading, translating, and delivering the data to the model in a format that it can receive, interpret, and dynamically display. The values within the model are refreshed within a timeframe compatible with the frequency at which the data is generated, ensuring a synchronised and dynamic representation.

This was achievable through the use of the game engine's programable capabilities, even though the data preparation was realised outside of the game engine, in future iterations these could be explored and packaged as a self contained application. It was identified that the refresh period of the Twents Waternet data in the website is for sensing data that is measured every hour, but this data is not always made available immediately as it is being revised and validated by the Twents Waternet team, this then generates a delay from the measurement to the moment it is downloaded. Additionally the density of sensors allows the coverage of the city of Enschede an its immediate surroundings, there are visually identifiable gaps that for these research had been filled through a basic interpolation method known as Delaunay Triangulation.

The requirements for sensor data while generating a Digital Twin are dependent upon the specific objectives of the Digital Twin itself. Consequently, the availability and density of sensors play a crucial role in determining which elements should be modelled, analysed, and interacted with. Ideally, a direct connection to the real-time data from sensors is desired, enabling timely updates on the measured elements. However, the timeframe for data production can vary depending on factors such as available storage capacity or the rate of changes occurring in the real-world element being measured. Furthermore, the physical limitations of the sensor's operating speed can also impose constraints on the available data. In cases where subtle and rapid changes pose risks to the project, a high refresh rate becomes essential. Conversely, if changes are more gradual but occur over an extended period, longer timeframes can be explored to capture the necessary data.

6.2. Development of a workflow for Digital Twin integration with real-time GWT sensor data

Stakeholder requirements for the Digital Twin system were comprehensively identified through a series of interviews and inspiration sessions involving a diverse group of multidisciplinary participants. The main objective was to explore feasible and pragmatic solutions that could be readily implemented with the available resources. Key considerations in these sessions included the need for real-time data integration, user-friendly interface, compatibility with existing software, methodologies, and scalability.

The following questions emerged as prominent concerns during these sessions:

1. Crowdedness in the subsurface and surface level: Identifying available space for water storage, heat networks, energy infrastructure, and tree roots.

2. Flood prediction: Can we accurately predict the locations and levels of floods resulting from intense rain or showers?

3. Groundwater pollution tracking: Is it possible to monitor and track the movement of groundwater pollution over time?

4. Tree planting optimisation: Which locations are best suited for planting specific types of trees, aligning with the Green Ambition Plan and Water and Climate Adaptation Plan?

5. Subsidence and sinkhole prediction: Can we monitor subsidence changes and predict potential sinkhole formation areas?

6. Assessment of intervention effectiveness: Can we demonstrate the effectiveness of drainage, infiltration, and water retention interventions?

7. Analysing attractiveness and avoidance: Which areas are more attractive to certain groups of people, and which areas are typically avoided?

8. Air pollution monitoring: How does air pollution change over time, and can we monitor its fluctuations?

9. Traffic flow management: Can we determine if the intended interventions to manage traffic flow have achieved their desired effects in terms of speed and intensity?

Even if not all these questions are relevant to the Digital Twin case, they served as possible exploration venues for the main topic of this research. The unused topics were then leveraged as the basis for the "Enschede LAB 2023," a collaborative initiative involving students from different universities to tackle specific challenges within the city. Throughout these discussions, a recurring and significant topic emerged: water management, particularly groundwater dynamics. As a result, the focus was primarily directed towards addressing this aspect of the research. The key requirements for the groundwater-related Digital Twin solution included adhering to the core concepts of Digital Twin methodology, employing open-source solutions, ensuring an intuitive user interface, and optimizing resource efficiency. The research's primary objective was then established to create a tool capable of monitoring, visualizing, and interacting with the groundwater table (GWT) in a robust and practical manner.

The workflow for Digital Twin integration involved data preprocessing, 3D modelling, real-time data acquisition, and visualisation. The research uses a range of software, tools and concepts including QGIS, CityEngine for modelling and geolocation concepts; Python and Google Colab for data preparation and management; the game engine Unity for visualisation, interaction, analysis and actions catalyser purposes. Additionally the use of ifttt.com for Internet automation and integration web applications was selected to control a physical water pump in the real world. For reference, see: Figure 8. Implementation of conceptual model of the working Digital Twin, forecast generation and its effect in the real-world.

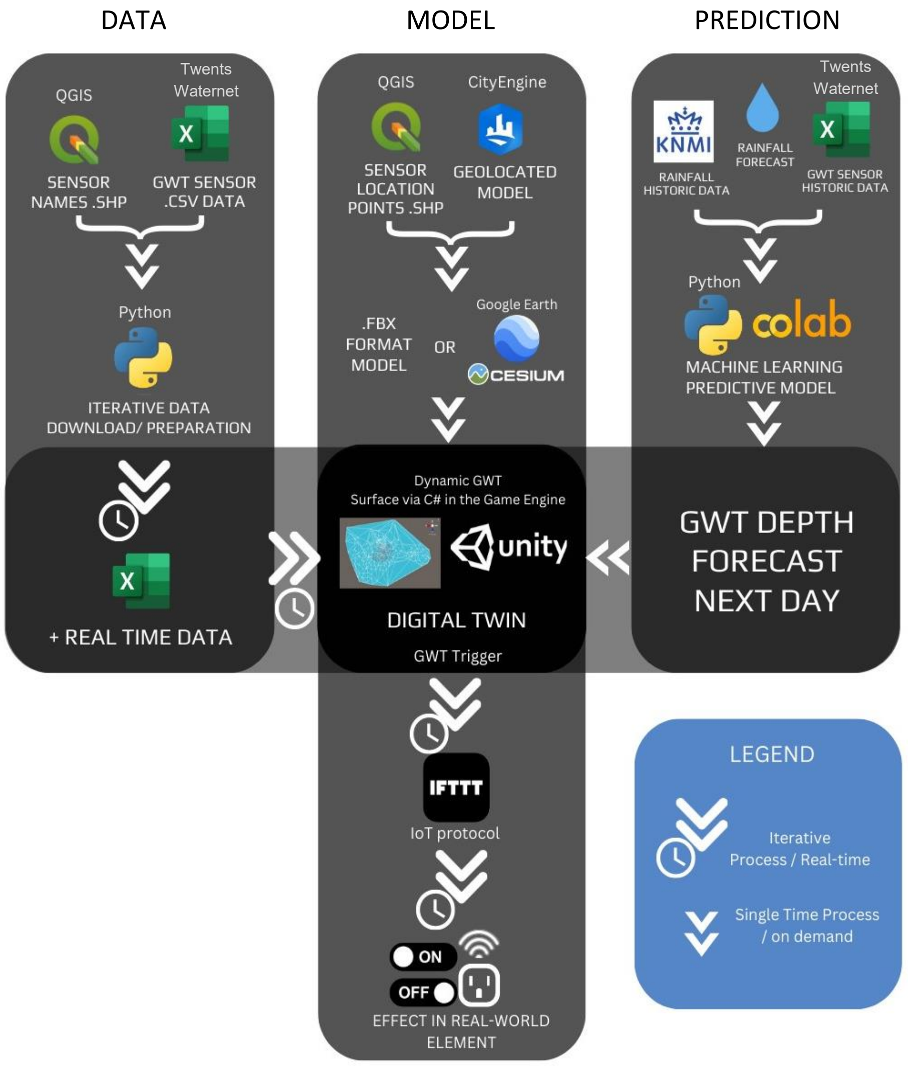


Figure 8. Implementation of conceptual model of the working Digital Twin, forecast generation and its effect in the real-world.

6.2.1. Sequential Code Development and Integration for Digital Twin Creation

Creating a functional digital twin, a dynamic virtual representation of a real-world system, requires a systematic and coordinated approach. The development of such a sophisticated model entails the integration of various codes, each designed to handle specific tasks such as data collection, processing, visualisation, prediction, and interaction. This section serves as a comprehensive guide, detailing the precise order and interdependencies of the codes necessary to construct an operational digital twin. From data collection and preprocessing to predictive modelling and interactive visualisation, each step is outlined to explain how the individual codes collectively contribute to the creation of a powerful, real-time monitoring and interaction platform. The order of using the codes is crucial for building the digital twin, as each code serves a specific purpose and builds upon the functionalities of the previous ones. It is also important to make an emphasis on the detail that even if game engines have the capabilities to generate a Digital Twin, they do not come prebuilt with out-of-the-box functions or tools pre-programmed for these purposes. Therefore, most of the interactions that are needed to integrate real-time data with the digital model and its interactions have to be crafted and programmed in order to achieve the desired outcome. As these integrations and interactions vary depending on the model, data and objectives there is no established procedure to tackle the workings of each part of the framework and a logical path had to be generated as specific needs came into view. The following show the sequence on how the different codes were developed:

1. Annex B: Data Collection Code (Python):
    - This code downloads raw data from a specific URL and processes it into CSV files.
    - It's the foundation, collecting the essential data required for further processing and analysis.
2. Annex C: Historic Data Preparation Code (Python):
    - This script takes raw data and extracts relevant information based on a specified date and time.
    - It prepares data for historical analysis, creating a refined dataset for historic representation.
3. Annex D: Training Data Preparation:
    - This code processes the downloaded CSV files from Annex B, ensuring consistent formatting of date-time data.
    - It's necessary for accurate training of machine learning models, which require consistent input data.

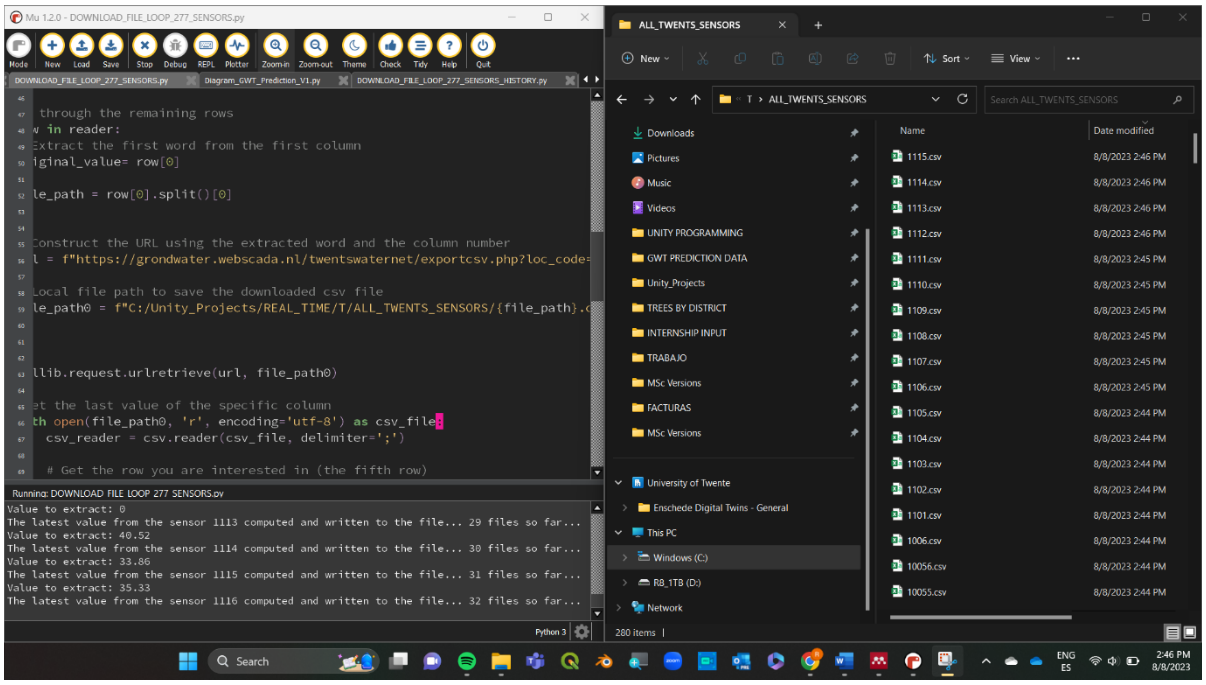
4. Annex E: Join GWT Data with Rainfall Data (RH in mm):
    - This script combines groundwater data with rainfall data, enriching the dataset with additional environmental context.
    - It enhances the dataset quality, enabling more accurate predictions and analysis.
5. Annex F: Real-time GWT Mesh Generator Code – Button A (C# - Unity):
    - This Unity script generates a 3D mesh based on real-time data.
    - It requires access to the real-time data generated by the Python code (Annex B).
6. Annex G: Historic Data Code - Button B and C (C# Unity):
    - This Unity script manages the dynamic representation of historical data using a 3D mesh.
    - It's dependent on the availability of the processed historical data from Annex C.
7. Annex H: Surface Collision Code (C# -Unity):
    - This Unity script creates interactive behaviour in response to colliders.
    - It depends on the mesh generated by the previous Unity script (Annex I) and requires interaction with the real-time data (Annex B).
8. Annex I: Button Control (Buttons A, B, C) Code (C# -Unity):
    - This Unity script controls the switching between real-time and historical data views.
    - It acts as a user interface control, allowing the user to toggle between different data representations.
9. Annex J: Camera Toggle Code (C# -Unity):
    - This Unity script allows toggling between different camera views in the scene (ThirteeNov Coding Vlog 2019).
    - It can only work effectively if the camera views have been set up, which might involve configuring the cameras' positions and orientations based on their intended perspectives.
10. Annex K: Camera Controls Code (C# -Unity):
    - This Unity script enables camera movement and controls (Davis 2019).
    - It's the final layer that allows users to interactively view the digital twin from different perspectives.
11. Annex L: GWT Decision Tree Predictive Model Code (Google Colab) :
    - This code involves data preprocessing, machine learning model training, and prediction.
    - It builds on the cleaned and enriched dataset to create a predictive model for groundwater levels.

Each code builds on the outputs or functionalities of the previous ones. The order ensures that the data is collected, prepared, and analysed correctly, and the Unity components are built upon accurate and relevant data representations.

6.2.2. Iterative data download/preparation:

In order to iteratively download, organise, and store the GWT data generated by sensors located in the Twents Waternet website. A Python code was developed to work iteratively in real-time Annex B: Data Collection Code (Python).

This code reads sensor data from a CSV file containing the names of each sensor from the Twent Waternet; it uses the values from the first column to construct URLs, and downloads the corresponding CSV files from a website. It then extracts the latest value from these downloaded CSV files and stores it in a general consolidated CSV file by overwriting it or creating a new one if it is non-existing yet. The purpose of this code is to collect and process data from the available sensors for further analysis or visualisation. This process is set to be repeated for a set amount of time according to the framework's needs and limitations. As the data is updated every hour the process is set to be done every 20 minutes to avoid overloading the Twents Waternet server as the data so far is not generated in a faster manner. Therefore, as long as this code is running the consolidated file will always be up to date with the latest GWT data available (see Figure 9).



```
 through the remaining rows
w in reader:
Extract the first word from the first column
iginal_value= row[0]

le_path = row[0].split()[0]


Construct the URL using the extracted word and the column number
l = f"https://grondwater.webscada.nl/twentswaternet/exportcsv.php?loc_code=

Local file path to save the downloaded csv file
le_path0 = f"C:/Unity_Projects/REAL_TIME/T/ALL_TWENTS_SENSORS/{file_path}.c



llib.request.urlretrieve(url, file_path0)

et the last value of the specific column
th open(file_path0, 'r', encoding='utf-8') as csv_file:
  csv_reader = csv.reader(csv_file, delimiter=';')

  # Get the row you are interested in (the fifth row)
```

```
Value to extract: 0
The latest value from the sensor 1113 computed and written to the file... 29 files so far...
Value to extract: 40.52
The latest value from the sensor 1114 computed and written to the file... 30 files so far...
Value to extract: 33.86
The latest value from the sensor 1115 computed and written to the file... 31 files so far...
Value to extract: 35.33
The latest value from the sensor 1116 computed and written to the file... 32 files so far...
```

Name	Date modified
1115.csv	8/8/2023 2:46 PM
1114.csv	8/8/2023 2:46 PM
1113.csv	8/8/2023 2:46 PM
1112.csv	8/8/2023 2:46 PM
1111.csv	8/8/2023 2:45 PM
1110.csv	8/8/2023 2:45 PM
1109.csv	8/8/2023 2:45 PM
1108.csv	8/8/2023 2:45 PM
1107.csv	8/8/2023 2:45 PM
1106.csv	8/8/2023 2:45 PM
1105.csv	8/8/2023 2:44 PM
1104.csv	8/8/2023 2:44 PM
1103.csv	8/8/2023 2:44 PM
1102.csv	8/8/2023 2:44 PM
1101.csv	8/8/2023 2:44 PM
1006.csv	8/8/2023 2:44 PM
10056.csv	8/8/2023 2:44 PM
10055.csv	8/8/2023 2:44 PM

Figure 9. Python code retrieving the latest available .CSV data from Twents Waternet

Addressing the need for historical analysis, the script Annex C: Historic Data Preparation Code (Python) refines the collected data. It takes a source CSV file, extracts specific values based on provided date and time criteria, and compiles this data into a result file. This refined dataset serves as the historical representation of the system, enabling the digital twin to accurately recreate past conditions and trends.

To facilitate machine learning model development, the code Annex D: Training Data Preparation focuses on data homogenisation. Building on the data collected in Annex B, the script processes and restructures the information into a consistent format. This ensures that the machine learning algorithms, which rely heavily on standardised data, can effectively recognise patterns and trends in the training phase. Environmental context plays a critical role in accurate predictions. The code

Annex E: Join GWT Data with Rainfall Data (RH in mm) combines groundwater data with rainfall data, enriching the dataset with an additional parameter. By linking groundwater level data to rainfall measurements, the digital twin gains insights into how external factors affect the system, enhancing its predictive capabilities.

6.2.3. Geolocated Models:

3D Located Buildings:

In order to effectively utilise the data continually generated by the aforementioned code, it becomes imperative to employ a model capable of reading, visualizing, and interacting with this data. Hence, for the present research, an optional step has been incorporated, wherein the creation of a 3D model of the study area's city is considered, leveraging the capabilities of CityEngine software. This 3D model is generated using building footprints in conjunction with a Digital Elevation Model (DEM) to extrude the building borders according to their respective heights. The CityEngine software operates on a rule-based approach, enabling modifications to an existing rule for buildings in Germany to adapt it to Dutch architecture, thereby ensuring a more accurate representation.

One of the notable advantages of employing parametric rule modelling is its efficiency in recreating multiple buildings simultaneously, facilitating the swift generation of a city model including textures within minutes. However, it is important to emphasise that this step remains optional. But if a pre-existing 3D model of the city buildings is already available for the study area, and it can be exported as an .FBX model, then it can be seamlessly integrated in the framework, as the calculation of the Groundwater Table (GWT) does not rely on the specific characteristics of the buildings.

Moreover, the level of detail in the 3D model can be tailored based on the specific requirements. For instance, the use of 3DBAG can provide scaled-down models with reduced intricacies, which are equally applicable for the objectives of the Digital Twin. Alternatively, the option of integrating Google Earth 3D models via the Google Earth API[2] can also be considered, provided it aligns with the research objectives of the Digital Twin implementation. This flexibility allows for the selection of an appropriate 3D model suited to the study's needs, accommodating various scenarios and refining the Digital Twin's fidelity.

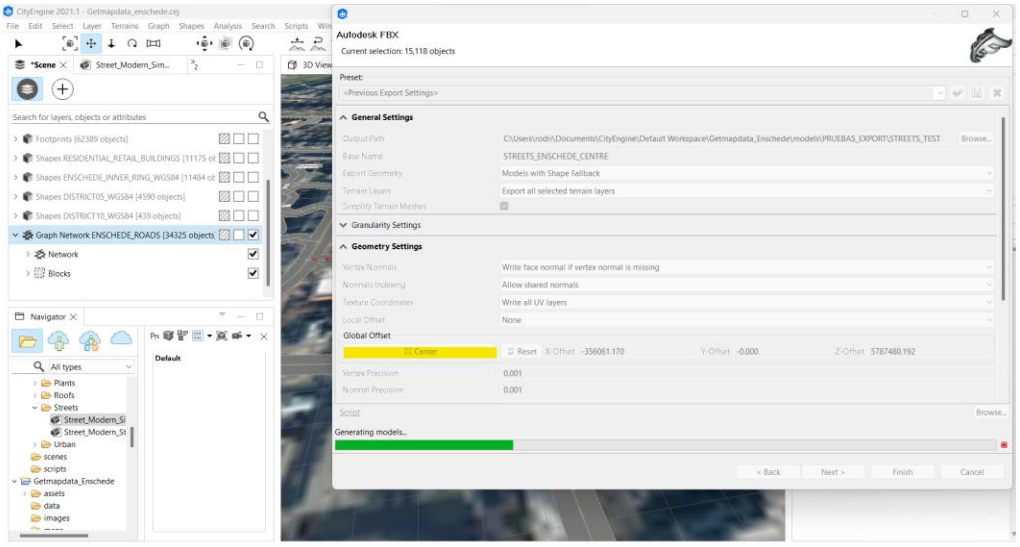
3D Surface:

The terrain surface plays a crucial role in the Digital Twin as it enables real-time checking for any interaction between the GWT and the terrain using hitboxes to trigger actions. To obtain this terrain representation, CityEngine offers the option to import a Data Terrain Model (DTM) file, which allows for the creation of a precise 3D depiction of the natural terrain devoid of buildings or trees. This accurate representation complements the GWT representation and the 3D buildings, making it ideal for the Digital Twin application.

[2] Example of Dynamic resolution and usage of Google Earth API in Unity (Morales, 2023), DYNAMIC RESOLUTION GOOGLE MAPS AND UNITY [Video] https://youtu.be/nGAsdC1QnM4.

For the obtained 3D models (buildings, surface, sensor points) the most reliable format that was identified to load directly in Unity was the .FBX as this format preserves important features of the original model, including textures, spatial reference within Unity, and the ability to treat each building as a separate entity.

A crucial consideration during the adjustment phase before exporting the 3D model as an FBX format is ensuring that the global offset is centred, simplifying the process of locating the model once it is loaded into Unity (see Figure 10). It is noteworthy that preserving georeferenced capabilities in Unity is not obligatory since the geolocation features are typically finalised and refined before the model is exported. Consequently, the files can be easily imported into Unity by simply dragging them into the project, enabling seamless integration of objects within a real-time rendering environment.


CityEngine 2021.1 - Getmapdata_enschede.cej
File Edit Select Layer Terrains Graph Shapes Analysis Search Scripts
*Scene
Street_Modern_Sim...
Search for layers, objects or attributes
Footprints [62389 objects]
Shapes RESIDENTIAL_RETAIL_BUILDINGS [11175 ol
Shapes ENSCHEDE_INNER_RING_WGS84 [11484 ot
Shapes DISTRICT05_WGS84 [4590 objects]
Shapes DISTRICT10_WGS84 [439 objects]
Graph Network ENSCHEDE_ROADS [34325 objects
Network
Blocks
Navigator
All types
Plants
Roofs
Streets
Street_Modern_Si
Street_Modern_St
Urban
scenes
scripts
Getmapdata_Enschede
assets
data
images
Default
3D View

Autodesk FBX
Current selection: 15,118 objects

Preset: <Previous Export Settings>

General Settings

Output Path: C:\Users\rodri\Documents\CityEngine\Default Workspace\Getmapdata_Enschede\models\PRUEBAS_EXPORT\STREETS_TEST
Base Name: STREETS_ENSCHEDE_CENTRE
Export Geometry: Models with Shape Fallback
Terrain Layers: Export all selected terrain layers
Simplify Terrain Meshes

Granularity Settings

Geometry Settings

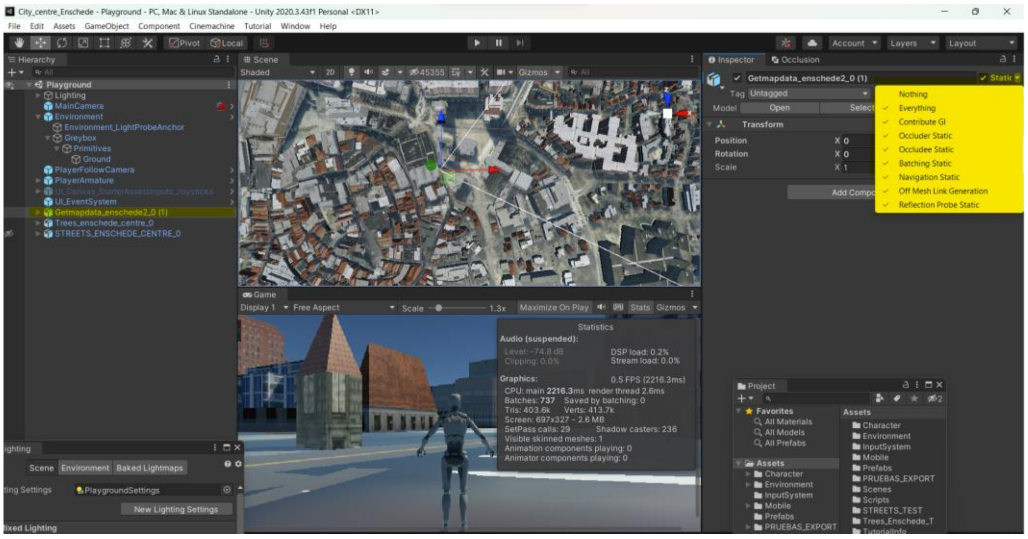
Vertex Normals: Write face normal if vertex normal is missing
Normals Indexing: Allow shared normals
Texture Coordinates: Write all UV layers
Local Offset: None
Global Offset: Center Reset X-Offset -356061.170 Y-Offset -0.000 Z-Offset 5787480.192
Vertex Precision: 0.001
Normal Precision: 0.001
Script

Generating models...

< Back Next > Finish Cancel

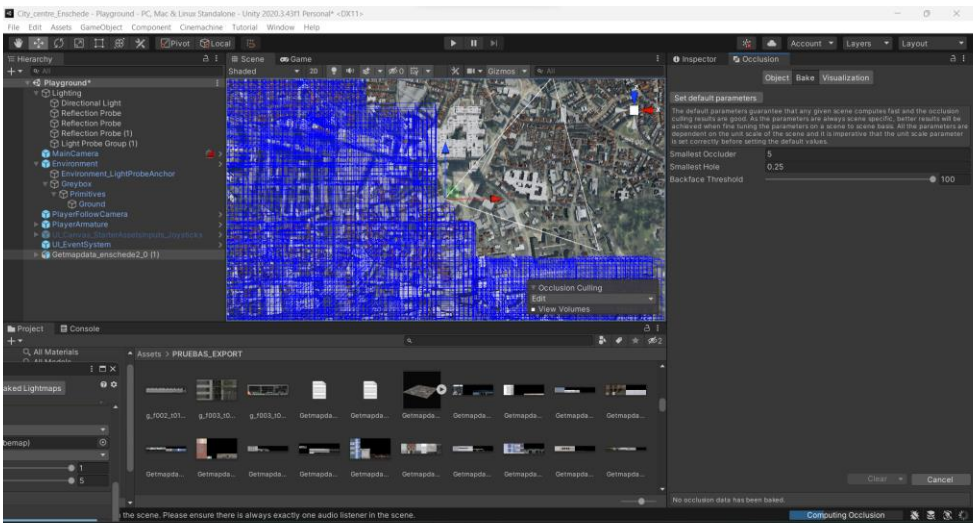
Figure 10. Export settings on an Autodesk FBX format from CityEngine.

Video-game development techniques were employed to optimise the rendering and visualisation of the entire model, objects that do not require movement, such as backgrounds and buildings, were designated as "static" objects in Unity (see Figure 11). This strategic approach enabled the implementation of Occlusion Culling on static objects. Occlusion culling generates data about your Scene in the Unity Editor, and then uses that data at runtime to determine what a Camera can see. The process of generating data is known as baking. (Unity 2022) By doing so, Unity only renders the portions of the model that are currently visible within the camera's field of view, automatically unloading any elements that lie beyond this visible range (see Figure 12). Notably, the real-time calculation of Occlusion Culling ensures that data is dynamically loaded and unloaded as the camera moves, facilitating seamless rendering and efficient resource utilisation during visualisation. After the application of Occlusion Culling, only the objects that are within the camera field of view will be rendered allowing Unity to process the model at higher frames per second (FPS) making the animation smooth, pleasant to the end user and less resource intensive. It is possible to see a jump in FPS from Figure 11 at a 0.5 FPS before applying occlusion culling to 93.2 FPS in Figure 13 after it was applied to the model, that represents approximately a 18540% in FPS performance boost for the model.



```
Statistics
Audio (suspended):
  Level: -74.8 dB                DSP load: 0.2%
  Clipping: 0.0%                 Stream load: 0.0%

Graphics:                        0.5 FPS (2216.3ms)
  CPU: main 2216.3ms  render thread 2.6ms
  Batches: 737    Saved by batching: 0
  Tris: 403.6k    Verts: 413.7k
  Screen: 697x327 - 2.6 MB
  SetPass calls: 29          Shadow casters: 236
  Visible skinned meshes: 1
  Animation components playing: 0
  Animator components playing: 0
```

Figure 11. Setting the model of the City Centre as a static object.

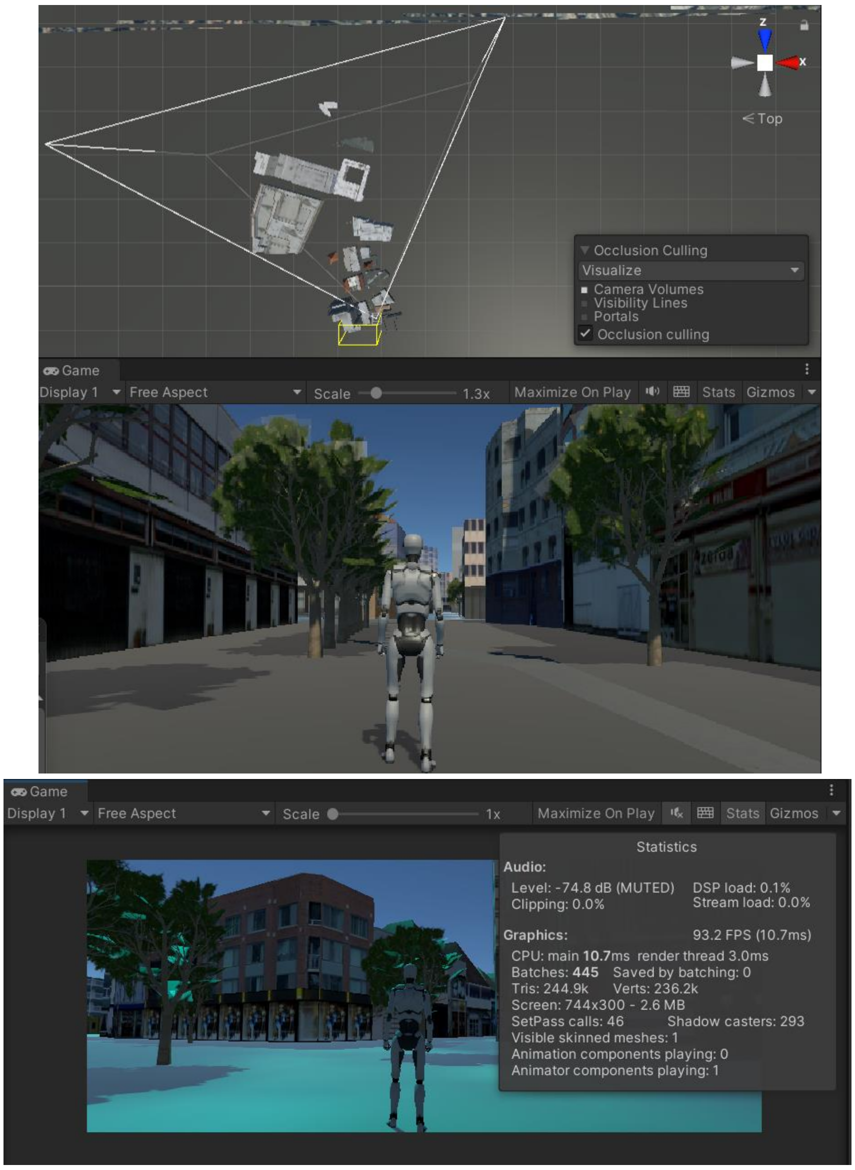


The default parameters guarantee that any given scene computes fast and the occlusion culling results are good. As the parameters are always scene specific, better results will be achieved when fine tuning the parameters on a scene to scene basis. All the parameters are dependent on the unit scale of the scene and it is imperative that the unit scale parameter is set correctly before setting the default values.

Smallest Occluder 5
Smallest Hole 0.25
Backface Threshold 100

No occlusion data has been baked.

Figure 12. Baking process for the application of Occlusion Culling.



```
Statistics
Audio:
  Level: -74.8 dB (MUTED)   DSP load: 0.1%
  Clipping: 0.0%            Stream load: 0.0%

Graphics:                   93.2 FPS (10.7ms)
  CPU: main 10.7ms  render thread 3.0ms
  Batches: 445   Saved by batching: 0
  Tris: 244.9k   Verts: 236.2k
  Screen: 744x300 - 2.6 MB
  SetPass calls: 46        Shadow casters: 293
  Visible skinned meshes: 1
  Animation components playing: 0
  Animator components playing: 1
```

Figure 13. Occlusion culling rendering only the objects within the field of view (lines in white).

6.3. Data availability

The case study area provided access to reliable and comprehensive data sources, including The Netherlands' Cadastre, Land Registry, and Mapping Agency.

The data used for the Digital Twin development was sourced from various providers, including Twents Waternet, Buienradar, KNMI, open data sources, private data sources, and historic GWT data. The IoT-based control of the water pump involved integrating real-world equipment data into the Digital Twin.

This real-time data serves as the basis for developing a digital twin (DT) in the research, as outlined in **the following table:**

NAME OF DATA FILE	SOURCE	OWNER SITE	RESTRICTIONS AND LICENCE	REAL TIME DATA	DATA FORMAT	PERSONAL INFO.
Open Street Maps buildings	Secondary	https://www.openstreetmap.org/export	Open data No restrictions	N	SHP file	N
Sensor locations	Secondary	https://grondwater.webscada.nl/twentswaternet/#	Not publicly downloadable	N	SHP file	N
Twents Waternet: Groundwater level	Secondary	https://grondwater.webscada.nl/twentswaternet/#	Open data No restrictions	Y	CSV file	N
KNMI	Secondary	https://knmy.readthedocs.io/en/latest/	Open data No restrictions	N	API recovered	N
Buien Radar	Secondary	https://www.buienradar.nl/	Open data No restrictions	N	Website	N

Table 6. Research Data Collection.

6.3.1. Available GWT Sensor Data in the city

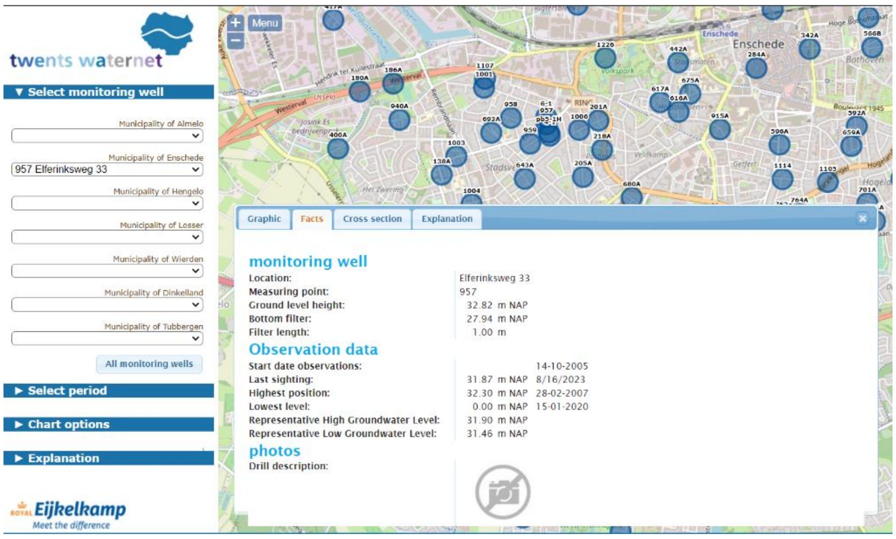
The municipality of Enschede has been working on analysing and preparing for different GWT scenarios, working on a redevelopment of affected areas with flooding and drought mitigation measures. Moreover, the collaboration between the Vechtstromen water board and the municipality of Enschede has resulted in the establishment of Twents waternet, a public website dedicated to smart wastewater processing in the region. Through this initiative, GWT sensors have been installed in Enschede, generating live and continuous measurements that are openly published on the website (Twents waternet 2022).

The availability of real-time data from sensors installed throughout the city has prompted organisations like the Municipality of Enschede to explore proactive solutions. Currently, the GWT data collected by sensors under houses in different neighbourhoods is used for periodic monitoring and manual water pumping in areas with high GWT levels. However, this process lacks automation, resulting in continuous and inefficient water pump operation even when the GWT is within acceptable limits. To address this issue, the Municipality seeks an effective approach to monitor the GWT using real-time data, enabling automated decision-making on when and where water should be pumped out. This would optimise energy and resource management and improve the overall efficiency of the pumping system.

The Twents Waternet dataset is a comprehensive collection of groundwater and environmental data from the Twente region. This dataset is fundamental for building a digital twin that models and predicts groundwater behaviour, interacts with users, and provides a visual representation of real-time and historical groundwater levels. The dataset is primarily collected through the Twents Waternet's monitoring systems, including various sensors distributed across the area (Twents waternet 2022).

Currently there are 277 sensors scattered across the City of Enschede, each sensor represents locations and points of interest within the Twente region. Each entry is associated with a unique identifier and a corresponding location. These identifiers serve as references to specific monitoring sites or neighbourhoods areas. Some of these sensors are inactive and some have different numbers but measure the same location. Therefore some cleaning and selection must be done when the data is downloaded to be used. The data is structured in a format that enables precise identification and categorisation of these locations for data collection and analysis purposes. This dataset serves as a crucial foundation for the proposed research framework, allowing the integration of environmental data from these points to build a comprehensive digital twin model for groundwater behaviour prediction and visualisation. The data included in the downloadable .CSV files includes the identification name/coding, the ground level height, the sise of the filter, the latest measurements, the highest/lowest measurement and dates when the observations took place. The data is measured hourly but goes through a validation process before it is made public to avoid misinformation (Twents waternet 2022).

The location of the sensors can be visualised in the Figure 4. Map of the study area and sensor location in the city of Enschede in The Netherlands.



monitoring well

Location:	Elferinksweg 33	
Measuring point:	957	
Ground level height:	32.82 m NAP	
Bottom filter:	27.94 m NAP	
Filter length:	1.00 m	

Observation data

Start date observations:		14-10-2005
Last sighting:	31.87 m NAP	8/16/2023
Highest position:	32.30 m NAP	28-02-2007
Lowest level:	0.00 m NAP	15-01-2020
Representative High Groundwater Level:	31.90 m NAP	
Representative Low Groundwater Level:	31.46 m NAP	

photos

Drill description:

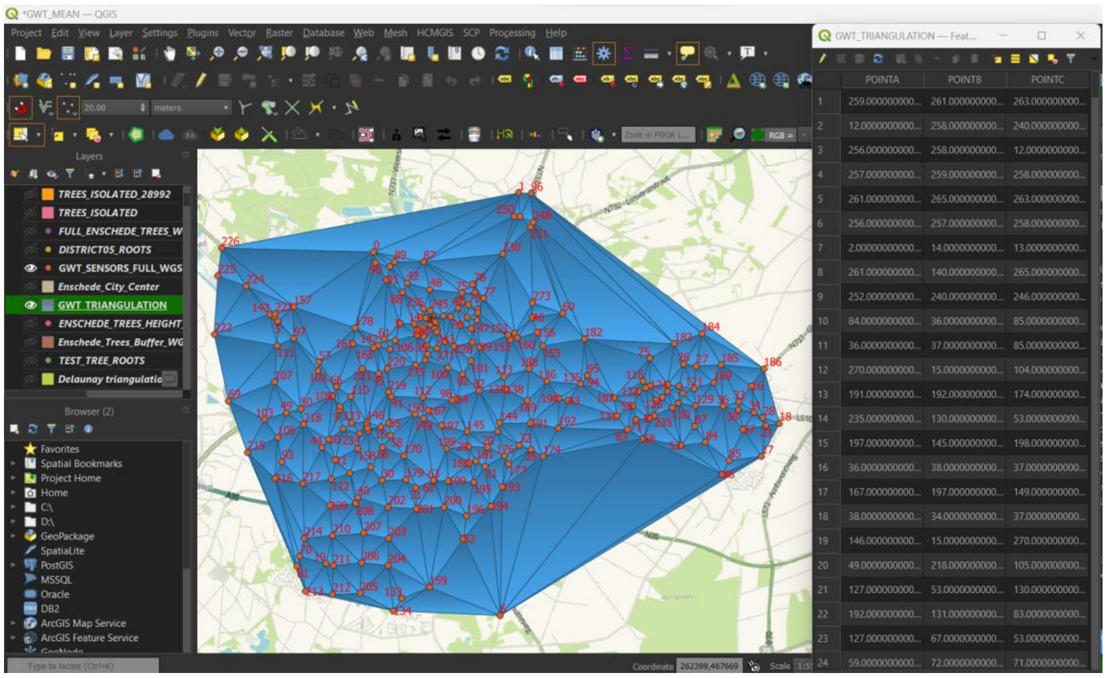
Figure 14. Data available per sensor on the Twents Waternet website.

6.4. Linking Real-Time Data with the Model

6.4.1. 3D Location of GWT Sensors and GWT Interpolation:

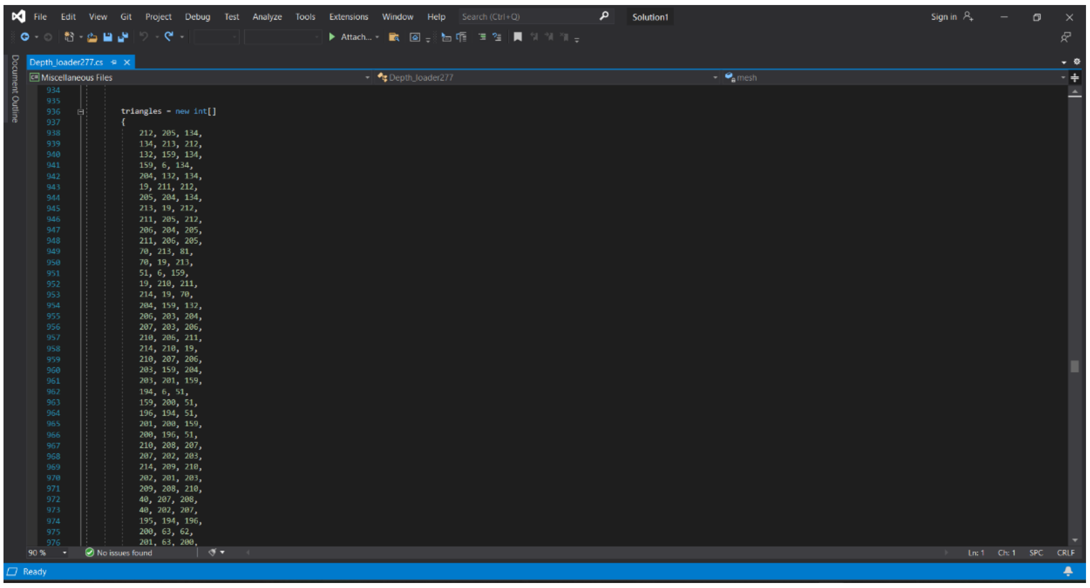
In order to generate a surface that represents the GWT levels throughout the study area, a way to interpolate the available data was needed. Inverse Distance Weighting (IDW) or Triangulated Irregular Network (TIN) interpolations where not used as the available Unity compatible options required the use of .RAW raster files and this turned out to be an uncommon format to work directly from a GIS perspective, but coded options can be considered for further developments of the framework. As the identified ways to do interpolations within Unity in real time were limited and time consuming, the usage of raster based interpolation was avoided in favour of a faster on-the-fly solution, a Delaunay Triangulation. The preparation for it had to be done only once as the X and Z positions of the sensors would not change. This also brings in the caveat that the interpolation will be less accurate as the triangulation assumes straight lines withing the available sensors positions.

In order to determine how to draw each triangle forming the GWT mesh in Unity, the Delaunay Triangulation tool in QGIS was employed. It's worth noting that the triangles must be drawn in a clockwise manner in Unity to ensure proper rendering. Fortunately, the Delaunay Triangulation in QGIS excludes repeated points with the same location, ensuring that the resulting polygon IDs match the original sensor IDs (see Figure 15). This is advantageous as it prevents the creation of duplicate surfaces for sensors at the same location and eliminates the need to reclassify IDs in a new order. Each point ID is grouped with two others to form a triangle, and the GWT's depth value serves as the Y-axis value during the triangle generation process in 3D. In order to generate the triangles in the Unity code it was important to each group of 3 points into a text format, therefore for the 270+ points it was easier to concatenate the values before copying them (see Figure 16).



	POINTA	POINTB	POINTC
1	259.000000000...	261.000000000...	263.000000000...
2	12.0000000000...	258.000000000...	240.000000000...
3	256.000000000...	258.000000000...	12.0000000000...
4	257.000000000...	259.000000000...	258.000000000...
5	261.000000000...	265.000000000...	263.000000000...
6	256.000000000...	257.000000000...	258.000000000...
7	2.00000000000...	14.0000000000...	13.0000000000...
8	261.000000000...	140.000000000...	265.000000000...
9	252.000000000...	240.000000000...	246.000000000...
10	84.0000000000...	36.0000000000...	85.0000000000...
11	36.0000000000...	37.0000000000...	85.0000000000...
12	270.000000000...	15.0000000000...	104.000000000...
13	191.000000000...	192.000000000...	174.000000000...
14	235.000000000...	130.000000000...	53.0000000000...
15	197.000000000...	145.000000000...	198.000000000...
16	36.0000000000...	38.0000000000...	37.0000000000...
17	167.000000000...	197.000000000...	149.000000000...
18	38.0000000000...	34.0000000000...	37.0000000000...
19	146.000000000...	15.0000000000...	270.000000000...
20	49.0000000000...	218.000000000...	105.000000000...
21	127.000000000...	53.0000000000...	130.000000000...
22	192.000000000...	131.000000000...	83.0000000000...
23	127.000000000...	67.0000000000...	53.0000000000...
24	59.0000000000...	72.0000000000...	71.0000000000...

Figure 15. Point configuration per Delaunay Triangle to be concatenated.



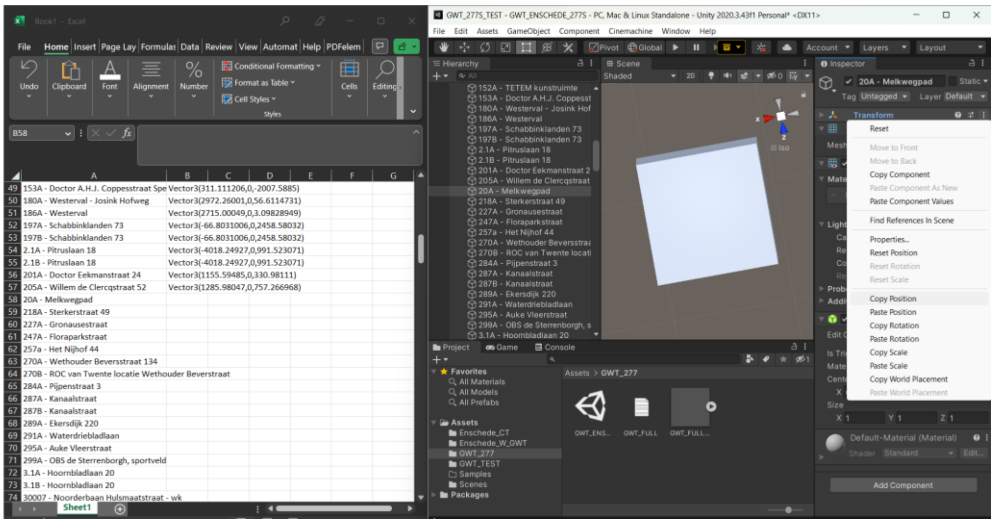
```
        triangles = new int[]
        {
            212, 205, 134,
            134, 213, 212,
            132, 159, 134,
            159, 6, 134,
            204, 132, 134,
            19, 211, 212,
            205, 204, 134,
            213, 19, 212,
            211, 205, 212,
            206, 204, 205,
            211, 206, 205,
            70, 213, 81,
            70, 19, 213,
            51, 6, 159,
            19, 210, 211,
            214, 19, 70,
            204, 159, 132,
            206, 203, 204,
            207, 203, 206,
            210, 206, 211,
            214, 210, 19,
            210, 207, 206,
            203, 159, 204,
            203, 201, 159,
            194, 6, 51,
            159, 200, 51,
            196, 194, 51,
            201, 200, 159,
            200, 196, 51,
            210, 208, 207,
            207, 202, 203,
            214, 209, 210,
            202, 201, 203,
            209, 208, 210,
            40, 207, 208,
            40, 202, 207,
            195, 194, 196,
            200, 63, 62,
            201, 63, 200,
```

Figure 16. Points sorted in rendering order in the mesh code.

The location of the sensors was important during the generation of the dynamic GWT inside the game engine. Therefore, as these points had to match the location of the surroundings of the study area, CityEngine was again used to convert the shape file points into 3D coordinates compatible with Unity. It is important to note that out of the box Unity does not have a coordinate reference system (CRS) as GIS tools or software do. Therefore the models are generated in a georeferenced environment in CityEngine and exported with a local reference, therefore as both the buildings and the sensors where created in CityEngine using the same reference system it was easier to make them match to each other when they were ported to Unity (see Figure 17 and Figure 18).

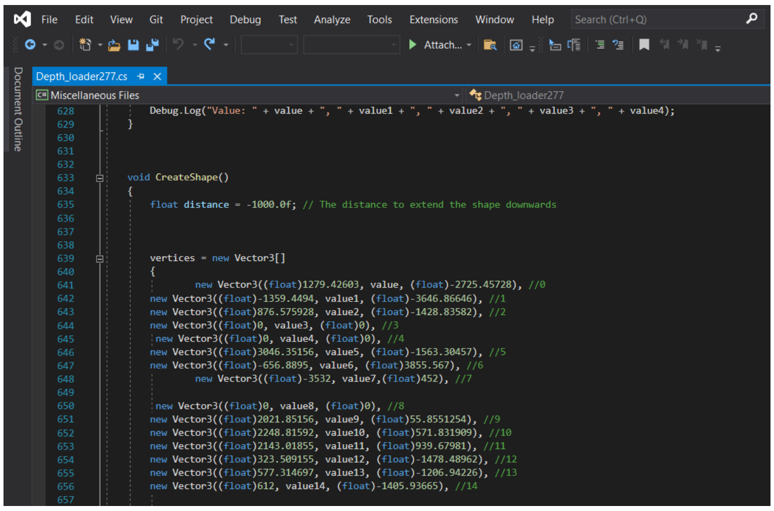
6.4.2. Real-time GWT Mesh Generation

Knowing the location in the local reference system of each sensor point in Unity and the order on which the triangulation had to be done, the programming of a code in C# that would dynamically generate in real time a mesh that represents the GWT level along the X, Y and Z axis was facilitated (see Annex F: Real-time GWT Mesh Generator Code – Button A (C# - Unity). This code developed in the Visual Studio capabilities of Unity. This C# script in Unity generates a 3D mesh using data from a CSV file. It reads values from the Consolidated CSV GWT file previously downloaded, stores them in variables, and uses them to create a mesh out of triangles, this mesh is then updated periodically. The script also creates spheres at the locations of the sensors right under the mesh as spheres contain hitboxes capabilities therefore later actions can be programmed whenever a sphere (that is moving in the Y axis at the exact GWT depth, if the spheres get in contact with the surface of the ground then it can trigger actions withing unity, this can be programmed to be messages or even signals that can be sent throughout the internet to communicate with IoT protocols. When the mesh generator code is operating in Unity it starts rendering the desired triangles in conjunction with the previously imported city model. This code works in a timed loop that can be set to different refreshing times (e.g. every 5 seconds or every hour) depending on the needs of the end user and the visualisation mode. The end result is an interactive 3D mesh that represents the GWT with dynamic values in the Unity environment (Figure 19 and Figure 20).



	A	B
49	153A - Doctor A.H.J. Coppesstraat Spe	Vector3(311.111206,0,-2007.5885)
50	180A - Westerval - Josink Hofweg	Vector3(2972.26001,0,56.6114731)
51	186A - Westerval	Vector3(2715.00049,0,3.09828949)
52	197A - Schabbinklanden 73	Vector3(-66.8031006,0,2458.58032)
53	197B - Schabbinklanden 73	Vector3(-66.8031006,0,2458.58032)
54	2.1A - Pitruslaan 18	Vector3(-4018.24927,0,991.523071)
55	2.1B - Pitruslaan 18	Vector3(-4018.24927,0,991.523071)
56	201A - Doctor Eekmanstraat 24	Vector3(1155.59485,0,330.98111)
57	205A - Willem de Clercqstraat 52	Vector3(1285.98047,0,757.266968)
58	20A - Melkwegpad	
59	218A - Sterkerstraat 49	
60	227A - Gronausestraat	
61	247A - Floraparkstraat	
62	257a - Het Nijhof 44	
63	270A - Wethouder Beversstraat 134	
64	270B - ROC van Twente locatie Wethouder Beverstraat	
65	284A - Pijpenstraat 3	
66	287A - Kanaalstraat	
67	287B - Kanaalstraat	
68	289A - Ekersdijk 220	
69	291A - Waterdriebladlaan	
70	295A - Auke Vleerstraat	
71	299A - OBS de Sterrenborgh, sportveld	
72	3.1A - Hoornbladlaan 20	
73	3.1B - Hoornbladlaan 20	
74	30007 - Noorderbaan Hulsmaatstraat - wk	

Figure 17. Obtaining the local coordinates of each vertex in Unity to identify the rendering points.



```
            Debug.Log("Value: " + value + ", " + value1 + ", " + value2 + ", " + value3 + ", " + value4);
        }


    void CreateShape()
    {
        float distance = -1000.0f; // The distance to extend the shape downwards



        vertices = new Vector3[]
        {
                new Vector3((float)1279.42603, value, (float)-2725.45728), //0
        new Vector3((float)-1359.4494, value1, (float)-3646.86646), //1
        new Vector3((float)876.575928, value2, (float)-1428.83582), //2
        new Vector3((float)0, value3, (float)0), //3
         new Vector3((float)0, value4, (float)0), //4
        new Vector3((float)3046.35156, value5, (float)-1563.30457), //5
        new Vector3((float)-656.8895, value6, (float)3855.567), //6
                new Vector3((float)-3532, value7,(float)452), //7

         new Vector3((float)0, value8, (float)0), //8
        new Vector3((float)2021.85156, value9, (float)55.8551254), //9
        new Vector3((float)2248.81592, value10, (float)571.831909), //10
        new Vector3((float)2143.01855, value11, (float)939.67981), //11
        new Vector3((float)323.509155, value12, (float)-1478.48962), //12
        new Vector3((float)577.314697, value13, (float)-1206.94226), //13
        new Vector3((float)612, value14, (float)-1405.93665), //14
```

Figure 18. Coordinates applied to the automatically updatable triangulated mesh code.

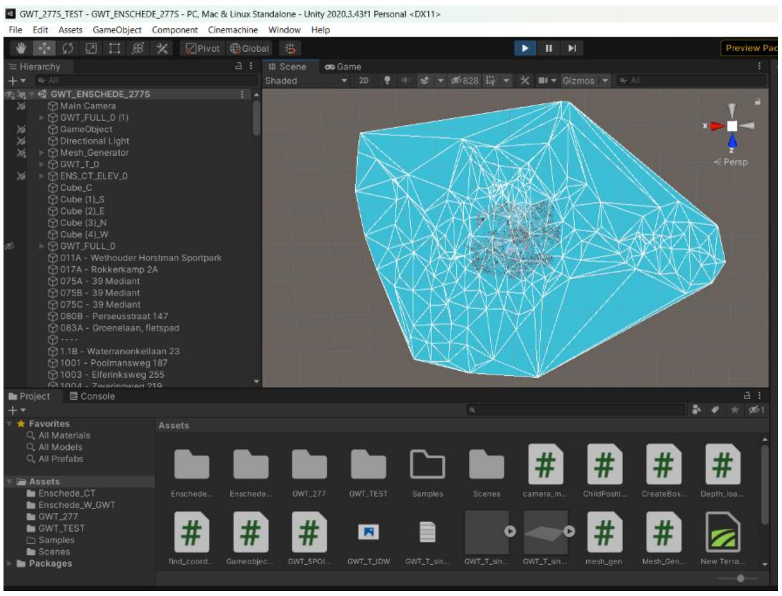

GWT_277S_TEST - GWT_ENSCHEDE_277S - PC, Mac & Linux Standalone - Unity 2020.3.43f1 Personal <DX11>
File Edit Assets GameObject Component Cinemachine Window Help
Hierarchy
Scene
Game
Project
Console
Assets

Figure 19. Real-time rendering of the GWT in Unity.

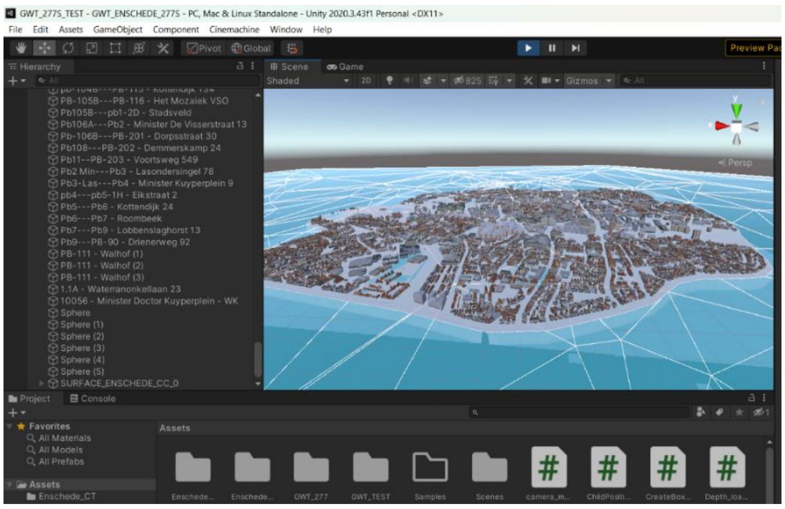

GWT_277S_TEST - GWT_ENSCHEDE_277S - PC, Mac & Linux Standalone - Unity 2020.3.43f1 Personal <DX11>
File Edit Assets GameObject Component Cinemachine Window Help
Pivot Global
Hierarchy
Scene Game
Shaded
Gizmos
Persp
PB-105B---PB-116 - Het Mozaiek VSO
Pb105B---pb1-2D - Stadsveld
Pb106A---Pb2 - Minister De Visserstraat 13
Pb-106B---PB-201 - Dorpsstraat 30
Pb108---PB-202 - Demmerskamp 24
Pb11--PB-203 - Voortsweg 549
Pb2 Min---Pb3 - Lasondersingel 78
Pb3-Las---Pb4 - Minister Kuyperplein 9
pb4---pb5-1H - Eikstraat 2
Pb5---Pb6 - Kottendijk 24
Pb6---Pb7 - Roombeek
Pb7---Pb9 - Lobbenslaghorst 13
Pb9---PB-90 - Drienerweg 92
PB-111 - Walhof (1)
PB-111 - Walhof (2)
PB-111 - Walhof (3)
1.1A - Waterranonkellaan 23
10056 - Minister Doctor Kuyperplein - WK
Sphere
Sphere (1)
Sphere (2)
Sphere (3)
Sphere (4)
Sphere (5)
SURFACE_ENSCHEDE_CC_0
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Enschede_CT
Assets
Enschede...
Enschede...
GWT_277
GWT_TEST
Samples
Scenes
camera_m...
ChildPositi...
CreateBox...
Depth_loa...

Figure 20. Each triangle is interconnected to their different X, Y, Z position in a 3D environment, automatically refreshed from the latest available data.

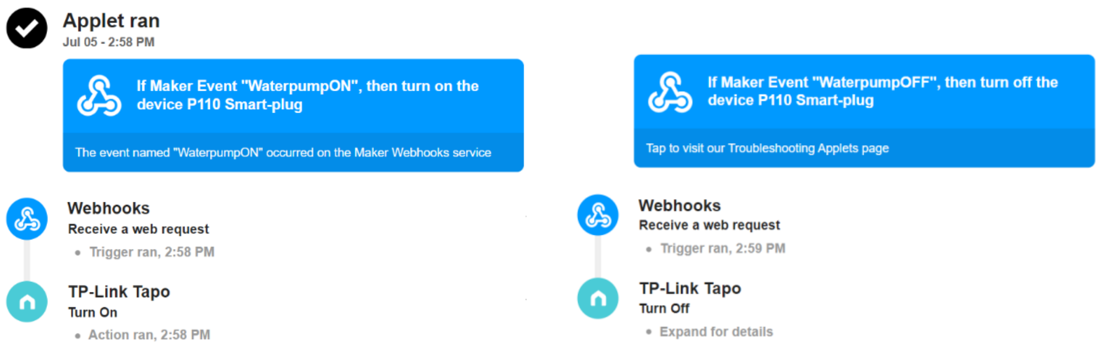
6.4.3. Loading Historic Data to the Model

Additionally a way to load historical data into the model was desired, therefore another code for this purpose was developed (see Annex G: Historic Data Code - Button B and C (C# Unity). As an extension of the previous Unity script, this code focuses on visual representation. By dynamically altering a 3D mesh based on historical data from Annex C: Historic Data Preparation Code (Python), the script creates an immersive visualisation of past conditions. It utilises the refined historical data to generate interactive mesh elements, allowing users to observe changes over time.

This C# script manages the creation and manipulation of a 3D mesh for the representation of the GWT. It reads data from the Historic CSV file to update values that influence the mesh's shape, then dynamically adjusts the mesh over time, creating spheres at certain vertices to visualise the changes. this script manipulates mesh shapes in a Unity scene. It reads values from a CSV file to drive changes in the mesh's appearance, animating it over time. The script demonstrates the dynamic modification of mesh geometry to create interactive visual effects in a game or interactive application. Additionally, this feature facilitates the loading of specific case scenarios into the model, enabling the deliberate testing of actions at predefined thresholds, independent of real-world metrics. This capability proves invaluable for scenario analysis and strategic planning across varying levels of intensity.

6.4.4. Simulation of scenarios for monitoring, management, and prediction of underground water levels

Different actions were identified for high and low GWT scenarios, including adjusting water extraction rates and implementing flood prevention measures. In the Digital Twin, a script was generated to react to actions that are activated depending on the GWT depth (see Annex H: Surface Collision Code (C# -Unity). For example the neutral state is when the GWT depth in all the city is below the surface, here only a monitoring action is performed and the visualisation of the GWT changes is possible. If the GWT rises above or touching the surface, then a maker event trigger is sent the IoT protocol (also known as applet) to the ifttt.com website and the linked action is performed. Therefore if the Maker Event happens then the command "WaterpumpON" is executed, which in this case is set to turn on the device P110 Smart-plug which consequently starts the water pump. On the contrary when the GWT lowers and passes from the state of touching the surface to not touch it anymore then the maker event trigger happens so the command "WaterpumpOFF" is executed, which in this case is set to turn off the device P110 Smart-plug. See the example of the activity log of the applet activating and deactivating the Smart-plug in Historic Data mode in Figure 21.


Applet ran
Jul 05 - 2:58 PM

If Maker Event "WaterpumpON", then turn on the device P110 Smart-plug

The event named "WaterpumpON" occurred on the Maker Webhooks service

Webhooks
Receive a web request
- Trigger ran, 2:58 PM

TP-Link Tapo
Turn On
- Action ran, 2:58 PM

If Maker Event "WaterpumpOFF", then turn off the device P110 Smart-plug

Tap to visit our Troubleshooting Applets page

Webhooks
Receive a web request
- Trigger ran, 2:59 PM

TP-Link Tapo
Turn Off
- Expand for details

Figure 21. Applet activity log showing the moments the WaterpumpON and the WaterpumpOFF.

While in the digital model, when a collider with a specific tag enters the trigger zone of the object (in this case the surface) it creates a cylinder and an animated object copy of a digital water pump to be shown in the point of collision as a beacon (as seen in Figure 22), this allows the user to know exactly where the water has entered the range to activate the water pump and facilitates finding the area in a large three-dimensional area from the likes of an entire city. This happens at the same time the command "WaterpumpON" is active. When the collider exits the trigger zone, it destroys the cylinder beacon and the animated water pump, as the command "WaterpumpOFF" activates. This was added to the model for the user to have feedback of the process that is being activated in an interactive/contextualised manner, therefore it gives the user a sense of location within the city (see Figure 23).

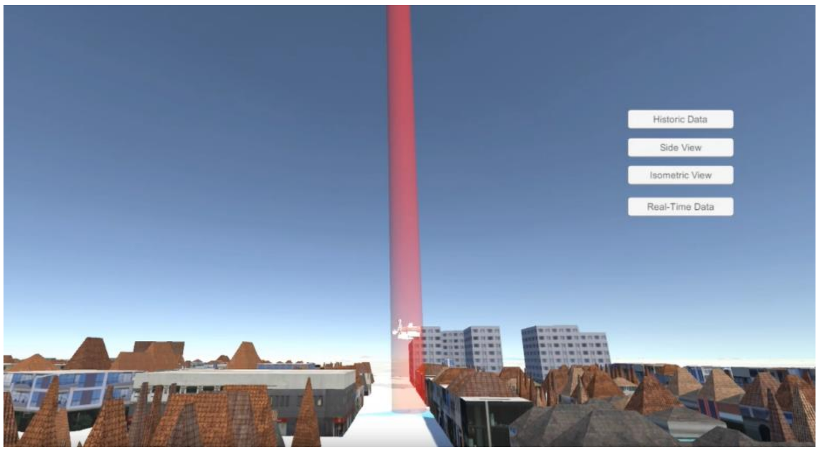


Figure 22. Cylindrical beacon seen from afar to identify the area of collision.

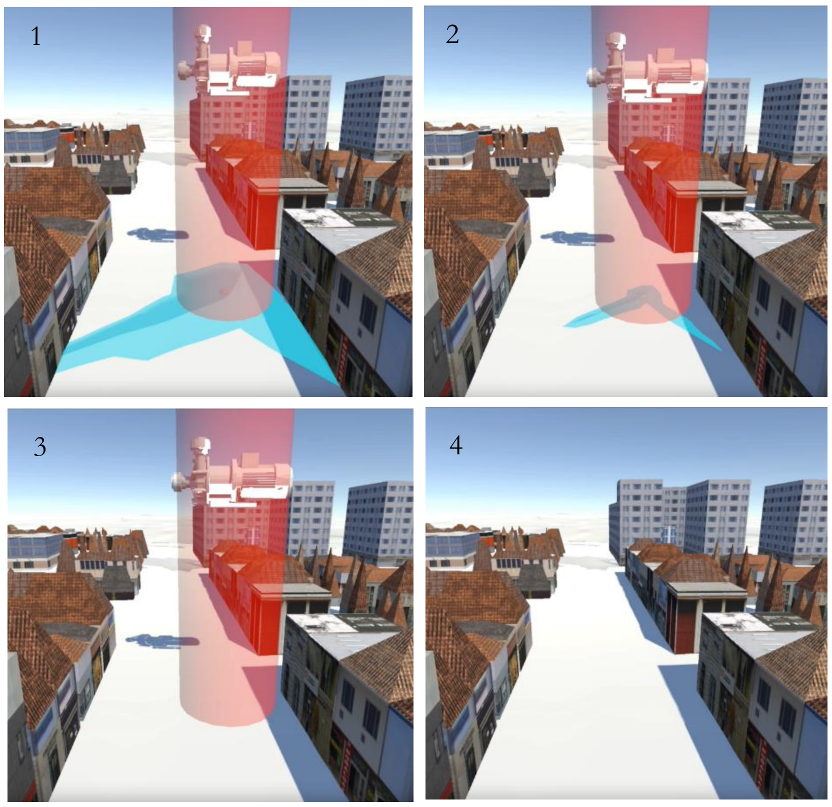
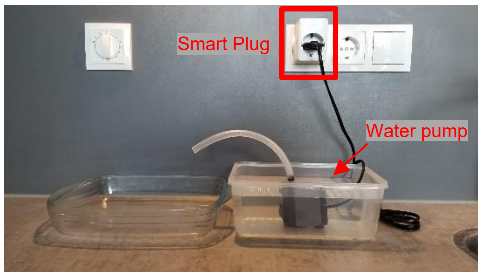


Figure 23. GWT drainage simulation, beacon and animated water pump behaviour.

6.4.5. Provision of Predictions to Help Decision Making in the Physical World:

The utilisation of Unity's collision and trigger capabilities has facilitated the development of a method that enables the transmission of commands through the internet to the website ifttt.com (ifthisthenthat). Leveraging this approach, specific actions can be triggered based on the collision of certain objects within the Unity environment. For instance, when the Groundwater Table (GWT) comes into contact with the surface of the city in non-waterbody areas, Unity generates a signal that initiates an action on the ifttt.com website. This action, in turn, activates a smart plug, which is internet-enabled and actively waiting for activation. The smart plug is directly connected to a small water pump situated in a controlled environment (see Figure 24 and Figure 24).

The scalability of this approach lies in the seamless integration of high capacity pumps deployed at strategic points throughout the city. These pumps are electrically operated and can be easily connected to the internet via Ethernet or WiFi. As a result, they can efficiently and remotely lower the GWT when triggered by the signal generated through the Unity simulation. Consequently, this scalable solution presents an effective means to regulate the GWT, as the process can be automated and remotely controlled through the internet, ensuring timely and responsive actions to address fluctuations in the water level. Additionally, the automated activation and deactivation of the water pump based on the GWT's contact with the city's surface optimise the pump's energy consumption and operational efficiency.



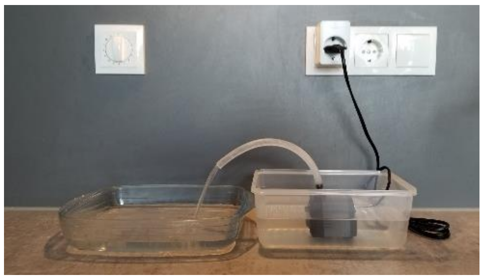


Figure 24 and Figure 25. Demonstrating the water pump in an ON/OFF state respectively.

Implementing water pumps to lower the Groundwater Table (GWT) not only presents a means of regulating and managing the water level but also offers several potential real-world benefits that can positively impact the city's water management and sustainability efforts. It is important to note the opportunity to repurpose the pumped water instead of discharging it into the drains. Instead, the water can be collected and stored in reservoirs strategically placed around the city. These reservoirs, which typically are filled up with rainwater, can benefit from this additional water source, especially during dry periods when rainwater collection may be insufficient. By collecting and storing the pumped water, the city can ensure a more reliable and consistent water supply for various purposes. One significant application of the stored water is for irrigation purposes, benefiting the city's green areas, trees, and vegetation. By utilizing this water for irrigation, the city can maintain healthier and more resilient landscapes while also conserving potable water resources. This aligns with sustainable practices and contributes to the overall environmental well-being of the urban ecosystem.

Moreover, the availability of stored water in reservoirs also presents a potential solution for scenarios where the GWT depths are too low, posing challenges for water availability. In such situations, the stored water can be utilised to augment the GWT levels, thus ensuring a more sustainable balance and preventing further depletion of groundwater resources.

From a broader perspective, the integration of water pumps, reservoirs, and the efficient management of collected water provides decision-makers with valuable data and tools for informed planning. The ability to control and regulate GWT levels offers opportunities for proactive water resource management, helping to avoid issues of water scarcity and improve the city's resilience to changing environmental conditions.

In summary, the inclusion of water pumps, coupled with the automatic monitoring and prediction possibilities to help with the collection and storage of pumped water. Therefore it goes beyond mere GWT depth lowering, as it presents a multifaceted approach that enhances water management, supports sustainable practices such as irrigation, and contributes to effective decision-making processes. Ultimately, this integrated system represents a step towards ensuring a more water-secure and resilient city, fostering a harmonious coexistence between urban development and environmental preservation.

For the current research, the data needs for monitoring the underground water level that can be used for predictions and decision making. These identified data categories collectively serve as pieces for the development of a base Digital Twin framework, offering a dynamic and multifaceted representation of the groundwater system. The necessary items for the presented framework are defined as follows:

- Real-Time Groundwater Data: Continuous and up-to-date data from underground water level sensors is crucial. Real-time data provides immediate insights into the current groundwater table (GWT) conditions, allowing for prompt decision-making and timely responses to any changes. This is the standard state of the framework working as the monitoring aspect.

- Historical Groundwater Data: Historical data covering past years, months, or days is essential for understanding the long-term behaviour and trends of the groundwater table. This data aids in identifying patterns, variations, and potential seasonal trends, enabling better predictions and planning for the future. Also allowed to test scenarios and visualise the behaviour of the GWT either by loading historic data or dummy data that gives an insight into what actions to take in specific cases.

- Predictive Groundwater Data: Predictive models generated through machine learning and other methods can provide forecasts of future groundwater levels. Having worst-case and best-case scenario predictions allows decision-makers to assess potential risks and plan appropriate management strategies. It has to be further developed and tested reviewing the forecasted values vs the real measurements in order to iteratively correct and mature its predictions.

- Terrain Data: Accurate representation of the natural terrain is crucial for the Digital Twin's accuracy. Terrain data helps in understanding the topography of the area, which can impact the movement of groundwater and its potential effects on the city's infrastructure. The masking or removal of water bodies should be considered to be avoided in future iterations as these can initiate unwanted triggers. Therefore, some debugging is still needed to be considered for the collision system.

- 3D Building Models: Building footprints in 3D models are essential for simulating the interaction between the underground water table and urban structures. Understanding how groundwater fluctuations can affect buildings and infrastructure is critical for urban planning and management decisions.

- Rainfall Forecasts: Rainfall data and forecasts provide insights into potential changes in the groundwater level due to precipitation. Incorporating rainfall forecasts helps in predicting how rain events may impact the GWT and can influence water management strategies.

- Interactive Visualisation Interface: An intuitive user interface within the game engine (Unity) is necessary to visualise and interact with the integrated data effectively. The interface should enable users to access real-time, historical, and predictive data easily, and switch between different views to gain a comprehensive understanding of the system.

- Geospatial Information: Geospatial information, such as the locations of sensors, buildings, and the terrain, is crucial for accurately positioning and integrating the different data components in the Digital Twin. Geospatial information ensures the data is aligned spatially, enabling precise simulations and analyses.

- IoT Protocols: The ability to send and receive data using IoT protocols is essential for connecting the Digital Twin with real-world sensor networks. This allows for bidirectional communication, enabling data synchronisation and enhancing the Digital Twin's real-time capabilities.

The integration and analyses of these data sources, allow the Digital Twin to become a valuable tool for supporting underground water level management decisions. The combination of real-time, historical, and predictive data, along with accurate representations of the terrain and 3D buildings, empowers decision-makers to understand the complexities of the groundwater system and make informed choices for sustainable water management. The framework's strength lies in its ability to mirror the complexity of the real world, providing decision-makers with the insights they need to manage this precious resource effectively in the dynamic urban landscape. By addressing these data aspects, the research drives groundwater management into an advance state characterised by foresight, adaptability, and sustainability.

6.4.6. Integration of overground, underground, and water sensor data

This integration was achieved through data fusion techniques and a centralised monitoring system. The Digital Twin based on a game engine (e.g., Unity) served as the central node that facilitated the interaction between different types of sensor data, including overground, underground, and water sensor data, as well as surface data such as the models of the city and rainfall forecasts. Unity's capabilities as a real-time 3D game engine played a crucial role in enabling this integration and interaction.

The Digital Twin acted as the central node using Unity during the following:

1. Real-time 3D Environment: Unity provided a powerful and interactive 3D environment where all the data components can be visualised and represented in real-time. The Digital Twin allowed the creation of a virtual representation of the study area, including the overground city structures, underground utilities, and the water table (GWT).

2. Data Integration: Unity's ability to handle multiple data formats and sources made it possible to import and integrate diverse sensor data into the Digital Twin. This included data from overground sensors (e.g., weather stations), underground sensors (e.g., utility monitoring systems), and water sensors (e.g., groundwater monitoring stations).

3. Real-time Processing: Unity's real-time processing capabilities enabled the continuous update and synchronisation of sensor data within the Digital Twin. As new data streamed in from different sensors, Unity processed and allowed the visualisation of the changes instantly, providing an up-to-date representation of the environment.

4. Interactive Simulations: By leveraging Unity's scripting capabilities, the Digital Twin was able to simulate interactions between different data components. For example, rainfall forecasts were dynamically applied to the model, affecting the simulated water table's behaviour in real-time.

5. IoT Protocols Integration: Unity's support for Internet of Things (IoT) protocols allowed bidirectional communication between the Digital Twin and physical sensors in the real world. This means that data can not only be gathered from the sensors but also sent back to them, enabling real-world control based on the Digital Twin's simulations and analysis.

6. Visualisation and Analysis: Unity's visualisation tools enhanced the understanding of complex data. Researchers were able to visualise the interactions between the overground structures, underground utilities, water table behaviour, and surface data (e.g., city models) simultaneously. This made easier to analyse and identify patterns, trends, and potential correlations between different data streams.

7. Decision Support System: The Digital Twin acted as a decision support system by providing a platform where researchers and stakeholders explored different scenarios and test the impact of changes in various parameters. This allowed for informed decision-making and planning based on the outcomes of the simulations.

The final interface of the developed framework allowed seamless interaction between the Digital Twin model and the physical world, enabling real-time monitoring and decision-making (see Figure 26). It is a user-friendly visualisation interface within the game engine (Unity). This interface features various buttons that can activate different modes, each catering to specific user needs. The buttons trigger corresponding codes to manage and display the data accordingly. Here's an explanation of the different buttons and their functionalities:

1. View Buttons: Additional view buttons are available to facilitate easy changes in the user's perspective of the Digital Twin model. For example, users can switch between different views, such as a top view, a side view, and an isometric view. These view buttons enhance the user's ability to explore the Digital Twin from various angles and gain a comprehensive understanding of the interactions between different components. The code to switch between cameras by pressing buttons Annex J: Camera Toggle Code (C# -Unity) was based on the example provided by (ThirteeNov Coding Vlog 2019) and the control Annex K: Camera Controls Code (C# -Unity) was developed by (Davis 2019).

2. Real-Time Data Button: This button will continuously load the latest sensor data into the Digital Twin, see Annex F: Real-time GWT Mesh Generator Code – Button A (C# - Unity). The user can access real-time information on underground water sensor data. As new data is collected from the physical sensors, it dynamically updates using the Annex B: Data Collection Code (Python)and this data is reflected in the Digital Twin's visualisation, providing an up-to-date representation of the environment. If pressed, the button calls the Annex I: Button Control (Buttons A, B, C) Code (C# -Unity) substituting the button code with the Annex G: Historic Data Code - Button B and C (C# Unity). Therefore this code when pressed changes the name of the button to "Historic Data" and prepares the code to load for the moment it is clicked once more.

3. Historic Data Button: By activating this button, users can load historical data from other time periods (e.g., past years, months, or days). See Annex G: Historic Data Code - Button B and C (C# Unity)This feature allows users to load historical behaviour of the groundwater table (GWT) allowing to understand its impact on different areas of the city in the past. It also enables the exploration of "what if" scenarios based on past data to assess how the GWT might have affected the city under different conditions. If pressed, the button calls the Annex I: Button Control (Buttons A, B, C) Code (C# -Unity) substituting the button code with the Annex F: Real-time GWT Mesh Generator Code – Button A (C# - Unity). Therefore this code when pressed changes the name of the button to "Predicted GWT Loader" and prepares the code to load for the moment it is clicked once more.

4. Predicted GWT Loader Button: This button uses the Annex G: Historic Data Code - Button B and C (C# Unity) to load the latest predictions generated with the machine learning code Annex L: GWT Decision Tree Predictive Model Code (Google Colab) . This is possible as the code looks for a .CSV file input that can be selected on the component that holds the code in Unity. These predictions represent worst-case scenarios for each predicted point, helping users prepare for potential adverse situations. By visualizing these predictions, decision-makers can plan and take proactive measures to mitigate risks associated with groundwater fluctuations.

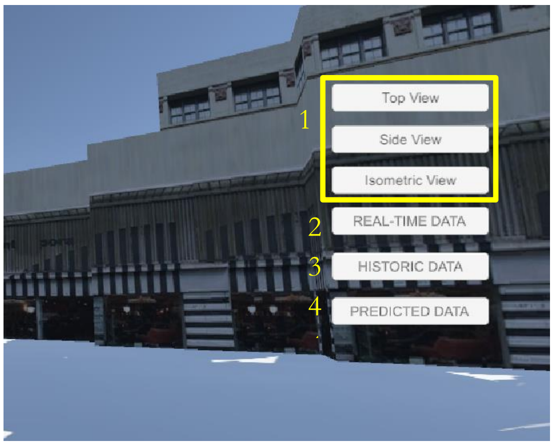


Figure 26. User interface with the coded buttons pertaining different actions.

Overall, the final interface provides a comprehensive and interactive platform that links and integrates physical world data from sensors with the digital model . Users have the flexibility to choose different modes to visualise and analyse real-time, historical, and predictive data. By providing access to various perspectives through view buttons, the interface empowers users to make informed decisions based on the simulation outcomes and enables them to understand the implications of the GWT on the city's infrastructure and urban planning.

6.4.7. Appropriate Models for GWT Predictions

Instead of a Physics-based model, such as groundwater flow models and hydrological simulations, a Machine Learning (ML) approach was utilised to predict GWT dynamics. Using the Figure 8 as reference to explain the method behind it. It starts by downloading the available data per sensor from the Twents Waternet website, this is done via a Python code (Annex B: Data Collection Code (Python). As this code generates the latest data available per sensor when it runs. The rationale behind the choice of the adoption of a ML approach over physics-based models for predicting groundwater table (GWT) dynamics within the study area generates from the need for a solution that can efficiently handle the substantial dataset generated by numerous sensors while providing quick insights into GWT changes. Additionally an idea of the amount of water that is drained daily in the city, either by natural means or by human interference was not publicly known, which is fundamental to generate a very basic physical model. On the other hand, the use of ML introduces an innovative way to tackle GWT prediction, demonstrating the adaptability of modern computational techniques in hydrological studies.

Consequently, a second Python code was developed (Annex D: Training Data Preparation) to prepare the data for the machine learning model, this code processes and formats the downloaded CSV files containing groundwater level data. The code reads groundwater depth data from one CSV file, processes the data, and saves it as a new CSV file with a consistent date format. It then checks and fixes the date format in all result files stored in the result directory. The final output is a set of CSV files with consistent date formatting and scaled depth values, ready for further analysis and use in the Digital Twin framework.

1. Create Directories and Set Variables: The code sets up the result directory to save the processed files. It also defines the CSV file path and the column number to be read from the input file.

2. Convert Data and Save Files: The code reads the input CSV file and processes each row. It extracts the first word from the first column and constructs a new file path. It then reads the corresponding CSV file, skipping the initial lines until it reaches the "datum" column. Afterward, it

converts the date and time formats and scales the depth value to meters. The processed rows are stored in a list, and a new result Consolidated Rainfall file is created with the converted data.

3. Fix Date Format: The code checks and fixes the date format for all the result files. Because some measurements come from dates as early as 1980 some of this values have different date formats. Therefore, it iterates through each file in the result directory and reads the data, checking for different date formats and converting them to a consistent format ("%d-%m-%Y %I:%M:%S %p"). It saves the updated data back to the respective files.

4. Printing Status Messages: Throughout the code, print statements are used to provide progress updates, indicating which files have been processed or fixed.

The data acquisition process, as explained in the section, features the importance of automation in data collection and preparation. The Python codes employed to extract and format data from the Twents Waternet website significantly reduce manual effort and ensure the latest available information is always utilised. This aspect showcases the role of scripting in streamlining data processes, a crucial element in ensuring the accuracy and reliability of subsequent ML model training.

After the data has been prepared to train the machine learning model, a phyton code that performs groundwater level prediction using a Decision Tree Regressor model is employed Annex L: GWT Decision Tree Predictive Model Code (Google Colab) . Based on hourly rainfall data (Rainfall Hourly - RH) obtained from the KNMI (Royal Netherlands Meteorological Institute) API, the code processes data from two CSV files, the one that contains the names of the GWT sensors and the Consolidated Rainfall file. This approach benefits from the relationship between hourly rainfall data and GWT levels. The significance of this approach lies in its ability to transform intricate hydrological processes into mathematical algorithms, offering predictions that are useful for urban planning and management.

The code then performs the following tasks:

1. Import Libraries: The code imports necessary libraries for data processing, machine learning, and working with CSV files.

1. Data Retrieval and Preparation: The code obtains rainfall data for the specified date range using the KNMI API and converts the RH values to a list, converting -1 values to 0.05 and scaling all other values by 0.1 to obtain the RH in millimetres.

2. Training the Model: The code reads groundwater level data from the Consolidated Rainfall .CSV file, prepares the input features (Date_Num, Hour_Num, RH(mm)), and the target variable (Depth(m)). It then splits the dataset into training and testing sets (80% - 20%) and trains the Decision Tree Regressor model.

3. Model Evaluation: The code evaluates the model's performance on the testing set using Mean Squared Error (MSE).

4. Generating Input Data for Prediction: The code generates input data for prediction by asking the user for the forecasted Rainfall for the 24 hours of the next day, creating a range of hours, and using these RH values for the calculations.

5. Predicting Groundwater Level: The code uses the trained model to predict the groundwater level (Depth(m)) for each hour of the specified date.

6. Print Predicted Depths: The code prints the predicted depths for each hour of the specified date. And saves the highest predicted GWT value per sensor file in a new consolidated file that is to be used in Unity to load the worst case scenario per sensor for the next day.

Overall, this code demonstrates how to use historical rainfall data to predict groundwater levels for a specific date using a Decision Tree Regressor model. It utilises machine learning techniques to establish a relationship between rainfall and groundwater levels, enabling prediction and understanding of potential flooding scenarios in a city. Moreover, the integration of data from different sources like the GWT sensors and KNMI's hourly rainfall data, within the ML framework showcases the potential of data fusion for enhancing prediction accuracy. This fusion of diverse datasets enriches the model's learning process, contributing to a more comprehensive understanding of GWT dynamics and potential scenarios.

Furthermore, the Python code's ability to standardise date formats across historical data demonstrates the software's capacity to handle data anomalies and inconsistencies. This becomes particularly relevant in long-term data collection efforts, where historical data from different sources might possess varying formats. This automatic standardisation process ensures the integrity of the data used for training and testing the ML model. In future iterations of the framework the Random Forest ML method can be considered and evaluated to identify if it is compatible with the available data in the study area it is applied.

6.5. Evaluation of the developed GWT monitoring system workflow

The monitoring system met the requirements of stakeholders, providing real-time data visualisation, predictions, and user-friendly interfaces.

The research encompassed a thorough consideration of diverse factors, among which was the selection of a suitable 3D modelling software. Ultimately, Unity, an open-source platform, was chosen for its emerging prominence within organisations, including municipalities in the Randstad region. Furthermore, successful integration of real-time data was achieved to establish a Digital Twin representation. Presently, the Enschede municipality is actively evaluating the incorporation of Unity into their Digital Twin processes, coinciding with similar initiatives like Nederland3D, which are keen on leveraging Unity for forthcoming undertakings.

As previously illustrated, the research was guided by the notion of employing the path of least resistance concerning available data and resources, to generate a decision-making framework. The framework's primary objective was to encourage city planners and multidisciplinary teams to comprehend the Groundwater Table's (GWT) dynamics and enable real-time action formulation based on its physical conditions. The adoption of the Digital Twin approach facilitated the realisation of this objective.

After the evaluation process was held it was identified that the monitoring framework successfully provided real-time data visualisation, allowing stakeholders to have up-to-date information on the Groundwater Table (GWT) and its fluctuations. The system was able to generate predictions based on historical data and user inputs, enabling stakeholders to anticipate potential changes in the GWT and plan accordingly. It also incorporated user-friendly interfaces, ensuring that stakeholders could interact with the data, visualise scenarios and access predictions through the use of buttons in Unity, fulfilling the stakeholders' requirement for an open-source tool with growing popularity among organisations like municipalities in the Randstad region. Additionally, the integration of real-time data into the Digital Twin was accomplished, providing a dynamic and accurate representation of the GWT and its interactions with the urban environment. Finally, the system effectively utilised available data and resources to develop a decision-making tool for city planners and multidisciplinary teams. This approach allowed stakeholders to make informed decisions based on real-time physical states of the GWT.

Stakeholder feedback from the showcased version of the working framework [3]highlighted the strengths of the developed workflow, such as the visual representation, control, editing capabilities, integration of smart data, its scalability and interoperability. Limitations were identified, including the need for physical connections with the water pump equipment throughout the city and interface enhancements.

While real-time data visualisation was implemented adequately, it can be hindered at times by the dependency on Twents Waternet's refresh rates. The data undergoes verification before being made available for download, which introduces delays. A direct connection between the sensors and the Digital Twin could streamline processes such as data preparation and formatting. Nonetheless, there is still ample room for improvement enhancing the user experience, as the current controls lack intuitiveness, necessitating a brief familiarisation period to navigate effectively in the 3D space environment.

The selection of historical data is working but can see improvements on the user interface selection. At the moment it is editable and configurable to the desired dates, but this is still done through data preparation that happens outside of the game engine and is important to develop further in future revisions. In the other hand the predictive capabilities are currently in their early stages and can be considered more as a proof of concept. The intention is to allow end users to develop and utilise their own predictive models. As prediction models perform optimally when trained on specific study areas, their effectiveness becomes subjective. The ideal predictive model should yield the most accurate outcomes tailored to the study area. Additionally, as for the data interoperability, it is important to research further methods to integrate the data preparation and machine learning operations within the game engine in order to be able to avoid human errors while handling the input and output data from these processes.
Finally, the scalability of the framework is considered to be possible by the stakeholders, as the next logical step is the implementation of real-size water pumps in test areas and study their effects in the GWT while being controlled through the established framework. Most importantly, in order for this to happen bugs, data interoperability, predictive models and interconnectivity with the new equipment should be refined before attempting to scale it up.

Additional features, such as integration with social media platforms for community engagement and advanced data analytics for proactive decision-making, were considered for future enhancements for the applicability of the monitoring system.

From the additional information provided, the following pending requirements can be identified:

- User-Friendly Interface: Simplifying the user interface to cater to stakeholders with limited coding experience or familiarity with controlling the Digital Twin using keyboard commands, virtual reality integration, and/or mixed reality implementation. These improvements would enhance accessibility and usability for city planners, university professors, and other stakeholders involved.

[3] First version of the working framework (Morales, 2023), WATERPUMP DRAIN ENSCHEDE [Video] https://youtu.be/i2R4tKhNhI8

- An improved and direct connection between the Digital Twin and the sensors should be further explored to facilitate the data integration process and avoid delays in data visualisation.

- Development of Predictive GWT Model: Although a predictive GWT model was provided within the research, it is still in the early stages of development. As a result, further refinement and validation of the model is required before it can be fully implemented and trusted for decision-making processes.

- Seamless Integration of Codes and Environments: Currently, some aspects of the system, such as data preparation and external CSV file handling, rely on tools outside of Unity, which might feel complicated for certain users. To improve cohesiveness and reduce reliance on external tools, additional research and development are needed to seamlessly integrate various components within the Unity environment.

Addressing these pending requirements would enhance the overall usability, reliability, and accessibility of the framework, making it more suitable for stakeholders with diverse technical backgrounds and ensuring its applicability for practical decision-making processes.

Strengths	Weaknesses
✓ **Successful development of monitoring system**	✕ Dependency on external refresh rates
✓ **Real-time data visualisation and predictions**	✕ Non-intuitive user interface controls
✓ **User-friendly interfaces and smart data integration**	✕ Reliance on external tools for data preparation
✓ **Scalability and interoperability**	✕ Early-stage predictive capabilities
✓ **Accurate portrayal of groundwater interactions**	✕ UI enhancements needed for historical data selection
✓ **Unity's open-source platform suitability**	✕ Integration challenges for predictive models
✓ **Alignment with emerging trends**	✕ Effectiveness of predictive models is context-dependent
✓ **Facilitation of groundwater understanding**	✕ Need for further integration of data handling in Unity

Opportunities	Threats
• **Development of comprehensive decision-making tool**	• Competition from alternative monitoring technologies
• **User interface enhancements for accessibility**	• Rapid technological advancements
• **Streamlined data preparation and handling in Unity**	• Regulatory changes affecting data availability
• **Direct connection between Digital Twin and sensors**	• Lack of stakeholder engagement and training
• **Integration with social media and advanced analytics**	

Table 7. SWOT analysis the overall usability, reliability, and accessibility of the framework.

These findings and limitations collectively provide insights into the accomplishments and challenges of implementing a Digital Twin-based groundwater monitoring system. The successful aspects highlight the system's potential, while the limitations indicate areas for improvement and further development.

6.5.1. Results Summary:

In general, the chapter presented a Digital Twin framework that managed Groundwater Table (GWT) behaviour in Enschede, Netherlands. The research explored various 3D modelling methods, with a focus on Digital Twins for real-time visualisation of GWT using game engines. Unity was selected for its suitability, and the integration process involved combining real-time sensor data, historical records, and predictive modelling. The framework's components and stages contributed to a comprehensive understanding of GWT dynamics and its interaction with the urban environment. Unity played a central role in integrating overground, underground, and water sensor data. Stakeholder feedback indicated successful achievement of real-time data visualisation, predictive capabilities, and user-friendly interfaces. Areas for improvement were identified, including user interfaces, data integration, predictive models, and streamlining processes. Finally, the framework's potential was evident in its ability to enhance decision-making and contribute to urban resilience.

7. CONCLUSION AND RECOMMENDATIONS

The results of this research show that integrating an open gaming format platform allows the development of Digital Twin solutions for maintenance in the public space. It specifically shows that climate change-induced issues like drought and water ingress in buildings can be avoided with a GWT digital twin solution comprising an actuator, various data sources including real-time GWT data, and a 3D visualisation in an open-source environment. Furthermore, the 3D environment in the Digital Twin has proven to be an intuitive way to interact with real-time data visualisation, allowing for a better understanding of the behaviours of the GWT and enabling quick actions based on its position relative to the surface. Additionally, the exploration of the interaction between real-time GWT data and the generated digital model has allowed us to identify possible interactions for flood/drought simulation. Therefore, the implementation of fore-casts will serve to estimate thresholds that consider how close the GWT is to the surface and use it as a triggering value to activate monitoring system messages or actions, such as turning on/off a physical/digital pump. The selected values are to be considered and programmed in the 3D modelling software, using the GWT levels, elevation data, and weather forecasts as inputs to calculate and evaluate estimates for future events.

7.1. Findings, limitations and Recommendations

The findings of this research highlight the potential of integrating Digital Twins for public space maintenance and effectively manage issues induced by climate change, such as droughts and water ingress, thanks to the intuitive nature of the 3D environment in the Digital Twin that fosters better understanding and enables prompt actions based on the GWT's behaviour. Likewise, the study also identifies potential interactions for flood and drought simulations, paving the way for further exploration and development.

The integration of Digital Twins for dynamic GWT control marks a significant advancement in water management strategies. However, the challenge lies in seamlessly merging real-time data with the Digital

Twin concept, creating a dynamic digital replica that can interact with physical actuators. To address this challenge, the successful integration of real-time data with physical actuators within the Digital Twin framework was used, therefore demonstrating a compelling solution for optimizing GWT management. This innovation enhances cities' capacity to not only monitor but also swiftly respond to fluctuations in the GWT, thus fostering a more efficient and responsive water management approach.

While utilizing a 3D model and real-time data within an open-source environment for Digital Twin implementations has been successfully developed. The challenge of further harnessing the full potential of game engines, particularly Unity's real-time rendering and action capabilities, remains. To answer this challenge, the establishment of an effective approach integrating real-time data with a 3D model in Unity showcases its potential. Further exploration into the capabilities of game engines like Unity is suggested to pave the way for more advanced and comprehensive Digital Twin research in the future.

Moreover, the 3D environment within the Digital Twin emerges as an intuitively interactive platform for real-time data visualisation. Nonetheless, the challenge at hand involves enhancing the controls and navigation within this environment to ensure an improved user experience. Refining the controls and navigation systems of the existing 3D environment offers the possibility to provide an intuitive and user-friendly interface. By addressing this challenge, the Digital Twin's utility for real-time data interaction can be significantly enhanced.

Equally important, the implementation of Digital Twins are able to generate tangible impacts on the physical world, presenting potential solutions to real-world problems. Nevertheless, scaling this solution effectively and optimizing its performance, particularly while integrating suitable equipment, poses a challenge. In order to work on this, ongoing research and development efforts should concentrate on refining scalability and performance aspects. Ensuring robust data interoperability, predictive models, and seamless interconnectivity is essential for realizing the potential of Digital Twins in practical applications.

In order to ensure an accurate representation within the Digital Twin, the formulation of precise visualisation principles and intuitive user interfaces is vital. The challenge entails crafting visualisation principles and interface designs capable of accurately capturing complex interactions while feeling natural for the end user. In response, thorough research and exploration into visualisation principles and user interface design are essential to establish an authentic and user-friendly representation within the Digital Twin environment.

Comprehensive data integration within the Digital Twin centres upon accurate sensor and pump alignments. However, strategically placing sensors and pumps across urban areas for comprehensive monitoring and representation remains a challenge. In order to address this, the thorough compilation and modelling of water pump units, combined with a strategic analysis of sensor placement, are indispensable for achieving a comprehensive and accurate representation of data within the simulation and case scenarios in the Digital Twin.

The scripts within the proposed framework exhibit coding practices that offer room for enhancement in terms of maintainability and efficiency. Using arrays or lists for repetitive variables could make the code more concise and easier to manage. Also, if there's a large number of similar variables that could potentially be achieved considered using arrays or dictionaries to store them. Enhancing code readability and optimizing performance through debugging and targeted code refinement is imperative. In the same way, enhancing the Digital Twin's predictive functionalities to simulate and forecast groundwater fluctuations demands further development. Yet, the refinement of predictive models to precisely simulate GWT dynamics triggered by meteorological events remains as a important challenge. To address this, continuous research efforts should concentrate on refining predictive models, leveraging advanced data models and IoT technologies to amplify the Digital Twin's predictive capabilities, thereby supporting its capacity for reliable forecasting.

Therefore, a successful real-world implementation requires the optimisation of the framework's performance and scalability. However, refining performance, data interoperability, predictive models, and integration with new equipment poses a challenge. In response, focusing on refining data interoperability, predictive models, and interconnectivity while rigorously testing and optimizing the solution will ensure its effectiveness in practical scenarios, allowing its application in the real-world.

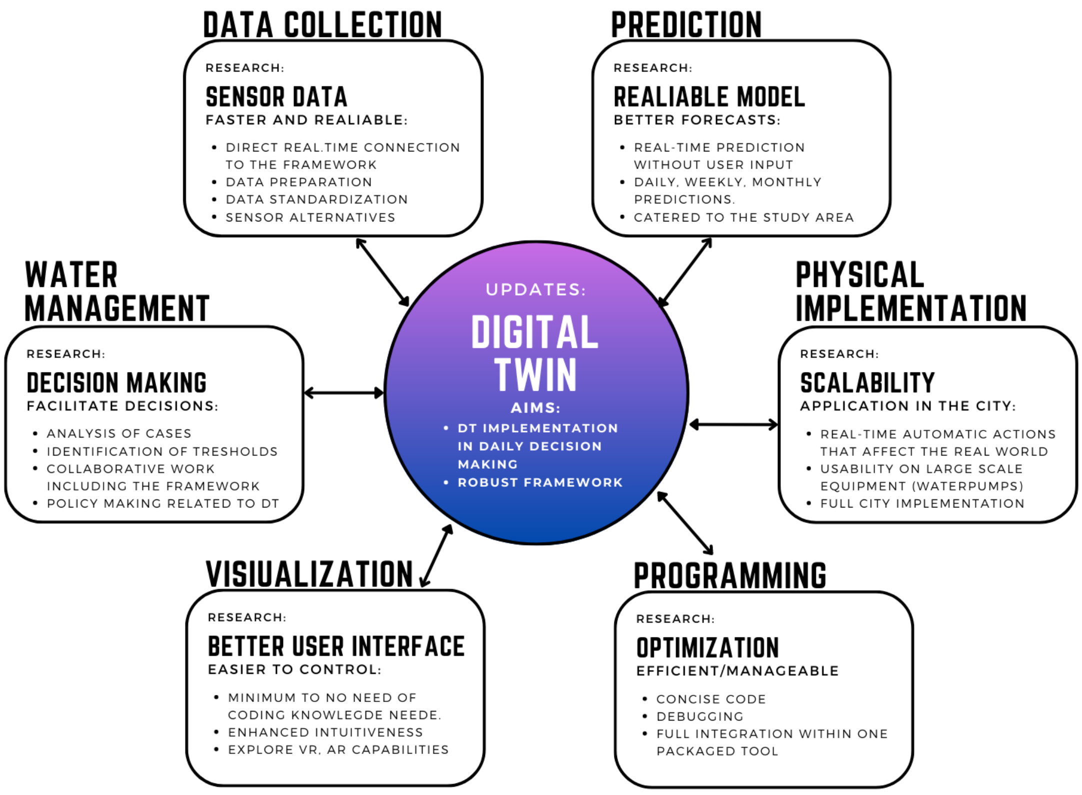
In order to make a substantial real-world impact, it is imperative to explore future applications of the developed solution. However, the challenge lies in identifying specific use cases and scenarios where the solution can be effectively employed. This can be attempted on future research and practical applications should be oriented towards identifying precise scenarios and contexts where the developed solution can be adapted to effectively tackle real-world challenges, thereby expanding its reach and potential.

Furthermore, the integration of Digital Twins into water management systems holds the promise of encouraging more resilient and sustainable cities. Nevertheless, the challenge at hand revolves around the continuous development and implementation of Digital Twins to ensure their effective contribution to urban water management. To tackle this, ongoing efforts in research, development, and practical implementation of Digital Twins in water management systems should be strengthened and considered into policy making decisions, ultimately leading to enhanced urban water management practices and more sustainable urban environments. Even though, the optimisation of the Digital Twin's performance can be achieved by leveraging faster hardware. However, the challenge lies in identifying suitable hardware upgrades and effectively integrating them. Therefore a strategic approach involves exploring and implementing appropriate hardware upgrades. By doing so, the performance of the Digital Twin can be significantly enhanced as it grows in capabilities, resulting in smoother operation, prevention of representation glitches, and the mitigation of data loss.

Additionally, the implementation of digital twin solutions for GWT management offers a range of advantages, but it requires careful optimisation. The challenge here involves further refining the implementation process and predictive capabilities to ensure optimal outcomes. To address this challenge, ongoing efforts should be directed towards meticulous improvement of the framework, including the predictive capabilities. This approach is key to obtain the maximum benefits of these solutions in GWT management and decision-making processes.

Finally, the proposed solution holds significant scalability potential and the prospect of automating water pump activation. However, this potential can be fully realised only after addressing pertinent challenges related to bugs, data interoperability, and predictive models (see

Figure 27). To meet this challenge, it is imperative to undertake comprehensive refinement of the solution. This rigorous refinement process should be tailored to address bugs, ensure data interoperability, and enhance predictive models. By doing so, the solution can be made robust enough to support successful automation of water pump activation on a larger scale.



DATA COLLECTION
RESEARCH:
SENSOR DATA
FASTER AND REALIABLE:
• DIRECT REAL.TIME CONNECTION TO THE FRAMEWORK
• DATA PREPARATION
• DATA STANDARDIZATION
• SENSOR ALTERNATIVES
PREDICTION
RESEARCH:
REALIABLE MODEL
BETTER FORECASTS:
• REAL-TIME PREDICTION WITHOUT USER INPUT
• DAILY, WEEKLY, MONTHLY PREDICTIONS.
• CATERED TO THE STUDY AREA
WATER MANAGEMENT
RESEARCH:
DECISION MAKING
FACILITATE DECISIONS:
• ANALYSIS OF CASES
• IDENTIFICATION OF TRESHOLDS
• COLLABORATIVE WORK INCLUDING THE FRAMEWORK
• POLICY MAKING RELATED TO DT
UPDATES:
DIGITAL TWIN
AIMS:
• DT IMPLEMENTATION IN DAILY DECISION MAKING
• ROBUST FRAMEWORK
PHYSICAL IMPLEMENTATION
RESEARCH:
SCALABILITY
APPLICATION IN THE CITY:
• REAL-TIME AUTOMATIC ACTIONS THAT AFFECT THE REAL WORLD
• USABILITY ON LARGE SCALE EQUIPMENT (WATERPUMPS)
• FULL CITY IMPLEMENTATION
VISUALIZATION
RESEARCH:
BETTER USER INTERFACE
EASIER TO CONTROL:
• MINIMUM TO NO NEED OF CODING KNOWLEGDE NEEDE.
• ENHANCED INTUITIVENESS
• EXPLORE VR, AR CAPABILITIES
PROGRAMMING
RESEARCH:
OPTIMIZATION
EFFICIENT/MANAGEABLE
• CONCISE CODE
• DEBUGGING
• FULL INTEGRATION WITHIN ONE PACKAGED TOOL

Figure 27. Visual summary of possible future framework enhancements.

LIST OF REFERENCES

Aguilar, Rosa. 2022. "CO-DESIGNING PLANNING SUPPORT TOOLS FOR STAKEHOLDER ENGAGEMENT IN COLLABORATIVE SPATIAL PLANNING PROCESSES."

Apple. 2009. "MacBook (13-Inch, Late 2009) - Technical Specifications (RO)." Retrieved August 24, 2023 (https://support.apple.com/kb/sp579?locale=ro_RO).

Bucchiarone, Antonio. 2022. "Virtual Reality & Intelligent Hardware Gamification and Virtual Reality for Digital Twin Learning and Training: Architecture and Challenges." 4(6):471–86. doi: 10.1016/j.vrih.2022.08.001.

Charest, Greg, and Mitch Rogers. 2020. "Data Exchange Mechanisms and Considerations | Enterprise Architecture." Retrieved August 15, 2023 (https://enterprisearchitecture.harvard.edu/data-exchange-mechanisms).

Chen, Kang, Xu Zhu, Burkay Anduv, Xinqiao Jin, and Zhimin Du. 2022. "Digital Twins Model and Its Updating Method for Heating, Ventilation and Air Conditioning System Using Broad Learning System Algorithm." *Energy* 251:124040. doi: 10.1016/j.energy.2022.124040.

Choo, Yeon Moon, Jong Gu Kim, Shang Ho Park, Tai Ho Choo, and Yeon Woong Choe. 2021. "Method for Operating Drainage Pump Stations Considering Downstream Water Level and Reduction in Urban River Flooding." *Water 2021, Vol. 13, Page 2741* 13(19):2741. doi: 10.3390/W13192741.

Chopine, Ami. 2011. "A History of Computer Graphics and Special Effects." *3D Art Essentials* 1–12. doi: 10.1016/B978-0-240-81471-1.10001-3.

Clausen, Christian Skafte Beck, Zheng Grace Ma, and Bo Nørregaard Jørgensen. 2022. "Can We Benefit from Game Engines to Develop Digital Twins for Planning the Deployment of Photovoltaics?" *Energy Informatics* 5(S4):42. doi: 10.1186/s42162-022-00222-7.

Davis, Ashley. 2019. "Unity FreeCam · GitHub." *Github*. Retrieved August 16, 2023 (https://gist.github.com/ashleydavis/f025c03a9221bc840a2b).

Digital Twin Geohub. 2022. "Modelling of the Urban and Rural Environements." Retrieved November 2, 2022 (https://www.utwente.nl/en/digital-society/research/themes/digital-twin-geohub/).

Edington, Lara, Nikolaos Dervilis, Anis Ben Abdessalem, and David Wagg. 2023. "A Time-Evolving Digital Twin Tool for Engineering Dynamics Applications." *Mechanical Systems and Signal Processing* 188:109971. doi: 10.1016/J.YMSSP.2022.109971.

Eicker, Ursula, Maryam Zirak, Nora Bartke, Laura Romero Rodríguez, and Volker Coors. 2018. "New 3D Model Based Urban Energy Simulation for Climate Protection Concepts." *Energy and Buildings* 163:79–91. doi: 10.1016/J.ENBUILD.2017.12.019.

Essam, Dina, Mohamed Ahmed, Abdou Abouelmagd, and Farouk Soliman. 2020. "Monitoring Temporal Variations in Groundwater Levels in Urban Areas Using Ground Penetrating Radar." *Science of The Total Environment* 703:134986. doi: 10.1016/J.SCITOTENV.2019.134986.

Gadea, C., B. Ionescu, and D. Ionescu. 2010. "Collaborative Web-Based Architecture for Real-Time Monitoring of Sensor Data." Pp. 52–58 in *2010 IEEE Workshop on Environmental Energy and Structural Monitoring Systems*. IEEE.

Gao, Yan, Haijiang Li, Guanyu Xiong, and Honghong Song. 2023. "AIoT-Informed Digital Twin Communication for Bridge Maintenance." *Automation in Construction* 150:104835. doi: 10.1016/j.autcon.2023.104835.

Gemeente Amsterdam. 2023. "GitHub - Amsterdam/Netherlands3D." Retrieved July 12, 2023 (https://github.com/Amsterdam/Netherlands3D).

Gigabyte. 2021. "AORUS 15P (RTX 30 Series) Key Features | Laptop - GIGABYTE Global." Retrieved August 24, 2023 (https://www.gigabyte.com/Laptop/AORUS-15P--RTX-30-Series#kf).

Gregory, Jason, and Richard Lemarchand. 2015. "Game Engine Architect." *CRC Press*.

Horton, Brandon K., Rajiv K. Kalia, Erick Moen, Aiichiro Nakano, Ken-ichi Nomura, Michael Qian, Priya Vashishta, and Anders Hafreager. 2019. "Game-Engine-Assisted Research Platform for Scientific Computing (GEARS) in Virtual Reality." *SoftwareX* 9:112–16. doi: 10.1016/j.softx.2019.01.009.

Huang, Justin, and Maya Cakmak. 2015. "Supporting Mental Model Accuracy in Trigger-Action Programming." Pp. 215–25 in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. New York, NY, USA: ACM.

ITC. 2021. "Offer of Notebooks | Ultrabook Lenovo Thinkpad L14 Gen4 (AMD) | Notebook Service

Centre." Retrieved August 24, 2023 (https://www.utwente.nl/en/nsc/offer-of-notebooks/ultra/).

Javaid, Mohd, Abid Haleem, and Rajiv Suman. 2023. "Digital Twin Applications toward Industry 4.0: A Review." *Cognitive Robotics* 3:71–92. doi: 10.1016/j.cogr.2023.04.003.

Kite-Powell, Jennifer. 2022. "Can Game Engines Help Digital Twins Visualise Data Better?" *Forbes*. Retrieved August 10, 2022 (https://www.forbes.com/sites/jenniferhicks/2022/06/22/can-game-engines-help-digital-twins-visualise-data-better/?sh=50fd09a1ca5e).

Klimaateffectatlas. 2022. "Waterlogging - Spatial Adaptation." Retrieved May 20, 2022 (https://klimaatadaptatienederland.nl/en/themes-sectors/themes/waterlogging/).

Lehtola, Ville V, Mila Koeva, Sander Oude Elberink, Paulo Raposo, Juho-Pekka Virtanen, Faridaddin Vahdatikhaki, and Simone Borsci. 2022. "Digital Twin of a City: Review of Technology Serving City Needs." *International Journal of Applied Earth Observations and Geoinformation* 114:102915. doi: 10.1016/j.jag.2022.102915.

Li, Wei, Yajian Wang, Hailu Yang, Zhoujing Ye, Pengpeng Li, Yang Aron Liu, and Linbing Wang. 2023. "Development of a Mixed Reality Method for Underground Pipelines in Digital Mechanics Experiments." *Tunnelling and Underground Space Technology* 132:104833. doi: 10.1016/j.tust.2022.104833.

Mi, Xianghang, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. "An Empirical Characterization of IFTTT." Pp. 398–404 in *Proceedings of the 2017 Internet Measurement Conference*. New York, NY, USA: ACM.

Moore, Ben. 2020. "IFTTT Review | PCMag." *PCMag*. Retrieved July 18, 2023 (https://www.pcmag.com/reviews/ifttt).

Nandi, Chandrakana, and Michael D. Ernst. 2016. "Automatic Trigger Generation for Rule-Based Smart Homes." Pp. 97–102 in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*. New York, NY, USA: ACM.

Normeninstituut Bomen. 2022. *Handboek Bomen 2022*.

Onile, Abiodun E., Ram Machlev, Eduard Petlenkov, Yoash Levron, and Juri Belikov. 2021. "Uses of the Digital Twins Concept for Energy Services, Intelligent Recommendation Systems, and Demand Side Management: A Review." *Energy Reports* 7:997–1015. doi: 10.1016/J.EGYR.2021.01.090.

Overijssel.nl. n.d. "Key Figures Province of Overijssel." Retrieved (https://www.overijssel.nl/over-overijssel/info-overijssel/kengetallen-provincie-overijssel/).

Pham, Quoc Bao, Manish Kumar, Fabio Di Nunno, Ahmed Elbeltagi, Francesco Granata, Abu Reza Md Towfiqul Islam, Swapan Talukdar, X. Cuong Nguyen, Ali Najah Ahmed, and Duong Tran Anh. 2022. "Groundwater Level Prediction Using Machine Learning Algorithms in a Drought-Prone Area." *Neural Computing and Applications* 34(13):10751–73. doi: 10.1007/S00521-022-07009-7/TABLES/4.

Program-Ace. 2021. "Unity vs. Unreal: Choosing the Best Engine | Program-Ace." *Program-Ace*. Retrieved July 17, 2023 (https://program-ace.com/blog/unity-vs-unreal/).

De Russis, Luigi, and Fulvio Corno. 2015. "HomeRules." Pp. 2109–14 in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM.

scikit. 2023a. "1.11. Ensemble Methods — Scikit-Learn 1.3.0 Documentation." Retrieved August 23, 2023 (https://scikit-learn.org/stable/modules/ensemble.html#random-forests).

scikit. 2023b. "Decision Tree Regression — Scikit-Learn 1.3.0 Documentation." Retrieved August 23, 2023 (https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#decision-tree-regression).

Slot, Aileen. 2019. "Zoveel Water Is Nodig Voor Jouw Grolsch-Biertje | Enschede Aan Zee | Tubantia.Nl." *Tubantia*. Retrieved December 6, 2022 (https://www.tubantia.nl/enschede/zoveel-water-is-nodig-voor-jouw-grolsch-biertje~a58aca89/).

ThirteeNov Coding Vlog. 2019. *How to Switch between Cameras in Unity3D*. Youtube.com.

Twents waternet. 2022. "Home - Twente Water Network." Retrieved January 18, 2022 (https://www.twentswaternet.nl/).

United Nations. 2020. "Sustainable Cities and Human Settlements | Department of Economic and Social Affairs." *United Nations*. Retrieved November 2, 2021 (https://sdgs.un.org/topics/sustainable-cities-and-human-settlements).

Unity. 2022. "Unity - Manual: Occlusion Culling." Retrieved August 8, 2023 (https://docs.unity3d.com/Manual/OcclusionCulling.html).

Wang, Zhoukai, Weina Jia, Kening Wang, Yichuan Wang, and Qiaozhi Hua. 2022. "Digital Twins Supported Equipment Maintenance Model in Intelligent Water Conservancy." *Computers and Electrical*

Engineering 101:108033. doi: 10.1016/J.COMPELECENG.2022.108033.

Xue, Fan, Weisheng Lu, Zhe Chen, and Christopher J. Webster. 2020. "From LiDAR Point Cloud towards Digital Twin City: Clustering City Objects Based on Gestalt Principles." *ISPRS Journal of Photogrammetry and Remote Sensing* 167:418–31. doi: 10.1016/J.ISPRSJPRS.2020.07.020.

Zhang, Weicun. 2023. "Compare Linear Regression, Decision Tree Regressor, and Random Forest Regressor Based on Python, a Restaurant Company on Kaggle as a Case." *BCP Business & Management* 36:322–29. doi: 10.54691/bcpbm.v36i.3449.

Zhou, Xiaoping, Haoran Li, Jia Wang, Jichao Zhao, Qingsheng Xie, Lei Li, Jiayin Liu, and Jun Yu. 2022. "CloudFAS: Cloud-Based Building Fire Alarm System Using Building Information Modelling." *Journal of Building Engineering* 53. doi: 10.1016/J.JOBE.2022.104571.

8. ANNEXES:

Annex A: Data management plan

Data	Storage Locations	Back-up location and frequency	Strategy to prevent unauthorised access to data during research	Strategy for pseudonymisation or anonymisation of data
Master files	UT cloud services. Safety and possibility for data sharing during the research	Google drive. All team members have real time access to all	View only option for external users. Access only by invitation	Data cleanup in line with Annex M: Ethical Considerations, Risks, and Contingencies
Copy of the data	Laptop local storage on each member's laptop	University of Twente P-drive	Create a password	Data cleanup in line with Annex M: Ethical Considerations, Risks, and Contingencies

Table 8. Organizing and Documenting Research Data

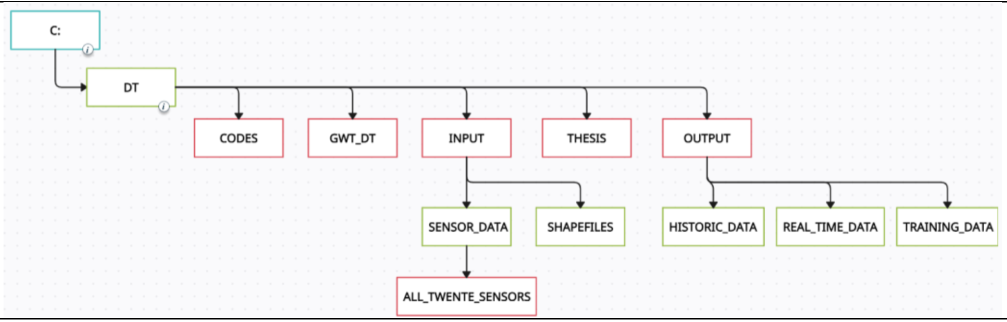
Name of main folder:	DT (Digital Twin)
Name of secondary and tertiary folders:	 C: DT CODES GWT_DT INPUT THESIS OUTPUT SENSOR_DATA SHAPEFILES HISTORIC_DATA REAL_TIME_DATA TRAINING_DATA ALL_TWENTE_SENSORS
Version control strategy:	Ver. 1.00: Initial draft; 26/10/2022 Ver. 1.10: Changes based on feedback; 00/11/2022 Ver. 2.00: First 3D camera test; 16/01/2023 Ver. 2.02: First person city test; 17/01/2023 Ver. 3.00: First real-time integration; 17/02/2023 Ver. 3.02: Surface integration; 20/04/2023 Ver. 3.04: Enschede third person test; 01/06/2023 Ver. 4.00: Enschede city Occlusion Culling; 28/06/2023 Ver. 5.00: GWT integration test; 16/08/2023 Ver. 5.02: City centre Enschede Sensors; 16/08/2023 Ver. 6.00: GWT 277 Sensors; 17/08/2023
Metadata standards used:	ISO 191155-1 metadata standards
Read-me file	Included in the DT folder

Table 9. Storage and Sharing of Research Data.

Annex B: Data Collection Code (Python)

```python
import csv
import time
import urllib.request
import os

# Local file path to save the result
result_file = "C:/Unity_Projects/REAL_TIME/T/Consolidated_V1.csv"

# Check if the directory exists
if not os.path.exists(os.path.dirname(result_file)):
    print("The directory does not exist. Creating the directory...")
    os.makedirs(os.path.dirname(result_file))

# Create the result file if it does not exist
if not os.path.exists(result_file):
    with open(result_file, 'w', newline='') as file:
        print("The result file does not exist. Creating the file...")
        pass

# Print the result file path for debugging
print("Result file path:", result_file)

# Set write permission on the file
os.chmod(result_file, 0o666)

# URL's to download the csv files

# Prompt the user to enter the path of the CSV file Change “file_to_read_path”
to your own computer path
csv_file_path = "C:/file_to_read_path/GWT_SENSOR_MEAN_VALUES.csv"
 #input("Enter the CSV file path: ")
column_number = 2
count=0

with open(csv_file_path, 'r', encoding='utf-8') as file:
    reader = csv.reader(file)

    # Skip the header row
    next(reader)

    # Loop through the remaining rows
    for row in reader:
        # Extract the first word from the first column
        original_value= row[0]

        file_path = row[0].split()[0]
        # Construct the URL using the extracted word and the column number
        url = 
f"https://grondwater.webscada.nl/twentswaternet/exportcsv.php?loc_code={file_p
ath}&loc_code_1=-1&loc_code_2={file_path}&loc_code_3=-1&loc_code_4=-
1&loc_code_5=-1&loc_code_6=-1&loc_code_7=-1&period=jaar&datumvan=09-02-
1800&datumtot=13-02-2050&HoogteUnit=cmNAP"

        # Local file path to save the downloaded csv file substitute 
“path_to_save_new_sensorfiles” with your own
        file_path0 = f”C:/path_to_save_new_sensorfiles/{file_path}.csv”



        urllib.request.urlretrieve(url, file_path0)

        #Get the last value of the specific column
        with open(file_path0, ‘r’, encoding=’utf-8’) as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=’;’)

            # Get the row you are interested in (the fifth row)
            row_to_extract = 4  # Rows are zero-indexed, so the fifth row has 
index 4

            for i in range(row_to_extract):
                next(csv_reader)

            # Extract the value in the second column of the fifth row
            try:
                value_to_extract = next(csv_reader)[1]

            except (ValueError, IndexError):
                # If there is no value or a non-numeric value in the specified 
cell, set it to 0
                value = 0

            # Print the extracted value for debugging
            print(“Value to extract:”, value_to_extract)


            # Skip the specified number of rows
            for i in range(12):

                # Try to skip the header row
                try:
                    next(csv_reader)
                except StopIteration:
                    # If the first row is empty, continue to the next file
                    continue

            # Get the last value of the specific column for each downloaded
file
            with open(file_path0, ‘r’) as file:
                reader = csv.reader(file, delimiter=’;’)
                last_value = None
                for row in reader:
                    try:
                        last_value = row[column_number]
                    except IndexError:
                        #print(“The row does not have the specified column.”)
                        continue

            #
            # Write the latest sensor depth value to the result file

            with open(result_file, ‘a’, newline=’’) as result:
                try:
                    writer = csv.writer(result)

                    writer.writerow([original_value, float(last_value)/100])
                except (ValueError, IndexError):
                    # If there is no value or a non-numeric value in the
specified cell, set it to 0
                    value = 0


        count=count+1
        print(f”The latest value from the sensor {file_path} computed and
written to the file… {count} files so far…”)


    print(“Downloading and processing of all CSV files completed.”)
```

The code is designed to download CSV files from a specific URL, process the data within these files, and then write the results to several CSV files, each one representing each available sensor.

1. File and Directory Handling:
   - The script sets up the paths for the result file and the directory in which it will be stored.
   - It checks if the directory exists and creates it if not.
   - It checks if the result file exists and creates it if not.

2. URL and CSV File Processing:
   - The script uses the specified path to a source CSV file. This file contains the names obtained from the GWT Sensors shapefile provided by Twents Waternet.

- It reads the source CSV file using the `csv.reader` with UTF8 encoding and skips the header row using `next(reader)`.

3. Loop Through CSV Rows:
   - The script then loops through the rows of the CSV file.
   - For each row, it extracts the first word from the first column (assuming that the CSV's first column contains a value that will be part of the URL).
   - It constructs a URL using the extracted value and a specified column number.
   - It uses `urllib.request.urlretrieve` to download the CSV from the constructed URL and save it to a local file path.

4. Extract Data from Downloaded CSVs:
   - The script then reads the downloaded CSV file using `csv.reader` and semicolon delimiter (based on the format of the CSV data).
   - It skips rows as needed (e.g., skips headers) to reach the desired row for data extraction.
   - It attempts to extract the value in the second column of the specified row.
   - It skips additional rows and extracts the last value of the specified column.

5. Write Processed Data to Result File:
   - The extracted original value and the last value from each file are written to the result file.
   - The result values are divided by 100 to convert from cm to meters before writing.
   - The script counts the processed files and prints a status message.

6. Completion Message:
   - Once all CSV files have been processed, the script prints a message indicating the completion of downloading and processing.

.

name
011A - Wethouder Horstman Sportpark
017A - Rokkerkamp 2A
075A - 39 Mediant
…

Table 10. Example of the file GWT_SENSOR_MEAN_VALUES.csv

Export meetgegevens			
Meetpunt	011A		
Locatie			
Maaiveldhoogte	0	(m)	
Bovenkant peilbuis	0	(m)	
Onderkant peilbuis	0	(m)	
Bovenkant filter	0	(m)	
Onderkant filter	0	(m)	
Periode	09-02-1800	t/m	13-02-2050
datum	tijd	meetwaarde	
18-08-1965	11:00	3088	
4/10/1965	11:00	3114	
18-10-1965	11:00	3102	
4/11/1965	11:00	3103	
18-11-1965	11:00	3098	
4/12/1965	11:00	3192	
…	…	…	

Table 11. Example of downloaded file for sensor 011A- Wethouder Horstman Sportpark

Annex C: Historic Data Preparation Code (Python)

```python
import csv
import os

# Local file path to save the result substitute "Specific_date_file_path" with
your own
result_file = "C:/Specific_date_file_path/Date_result.csv"

# Check if the directory exists
if not os.path.exists(os.path.dirname(result_file)):
    print("The directory does not exist. Creating the directory...")
    os.makedirs(os.path.dirname(result_file))

# Create the result file if it does not exist or clear it if it exists
if not os.path.exists(result_file):
    with open(result_file, 'w', newline='') as file:
        print("The result file does not exist. Creating the file...")
        pass
else:
    # Clear the result file by truncating it
    with open(result_file, 'w', newline='') as file:
        print("Clearing the result file...")

# Print the result file path for debugging
print("Result file path:", result_file)

# Set write permission on the file
os.chmod(result_file, 0o666)

# Prompt the user to enter the path of the CSV file substitute
“file_to_read_path” with the path leading to the file with the sensor names
csv_file_path = "C:/file_to_read_path/GWT_SENSOR_MEAN_VALUES.csv"
# input("Enter the CSV file path: ")
column_number = 2
count = 0

with open(csv_file_path, 'r', encoding='utf-8') as file:
    reader = csv.reader(file)

    # Skip the header row
    next(reader)

    # Loop through the remaining rows
    for row in reader:
        # Extract the first word from the first column
        original_value = row[0]
        file_path = row[0].split()[0]

        # Local file path to read the downloaded csv file substitute
“path_to_save_new_sensorfiles” with your own
        file_path0 = f"C:/path_to_save_new_sensorfiles/{file_path}.csv"

        # Get the last value of the specific column
        with open(file_path0, 'r', encoding='utf-8') as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=';')

            # Find the row that matches the specified date and time you can
choose a date and time that is available in your data
            date_to_match = "21-12-2016"
            time_to_match = "12:00"
            matching_row = None

            for row in csv_reader:
                if len(row) >= 3 and row[0] == date_to_match and row[1] ==
time_to_match:
                    matching_row = row
                    break

            # Extract the value from the 3rd column of the matching row
            if matching_row is not None:
                value_to_extract = float(matching_row[2])
            else:
                value_to_extract = 0

            # Print the extracted value for debugging
            print("Value to extract:", value_to_extract/100)

            # Write the extracted value to the result file
            with open(result_file, 'a', newline='') as result:
                writer = csv.writer(result)
                writer.writerow([original_value, value_to_extract/100])

        count += 1
        print(f"Value for {file_path} computed and written to file... {count}
files so far...")

print("Processing of all CSV files completed.")
```

This script processes a source CSV file, extracts specific values from corresponding CSV files based on a given date and time, and writes the extracted values to a result file. This file is later possible to be loaded as historic data in the Digital Twin.

1. Directory Handling and Prompt:
    - The script sets up the directory path where the result file will be saved.
    - It checks if the directory exists and creates it if not.
    - The user is prompted to enter the path of the source CSV file.

2. CSV Processing Loop (First Part):
    - The script reads the source CSV file using `csv.reader`.
    - It skips the header row using `next(reader)`.

3. Loop Through Source CSV Rows:
    - The script loops through the rows of the source CSV file.
    - For each row, it extracts the first word from the first column.

4. Get Last Value from Specific CSV:
    - It constructs the file path to read the downloaded CSV file.
    - The script reads the specified CSV file.
    - It searches for a row that matches a given date and time.
    - If a match is found, it extracts the value from the 3rd column of that row.

5. Write Extracted Values to Result File:
    - The script writes the extracted value along with the original value from the source CSV to the result file.

6. Completion Message (First Part):
    - The script counts the processed files and prints a status message.

7. Final Message:
    - The script prints a completion message for the entire process.

011A - Wethouder Horstman Sportpark	0
017A - Rokkerkamp 2A	47.68
075A - 39 Mediant	0
075B - 39 Mediant	0
075C - 39 Mediant	0
080B - Perseusstraat 147	0
083A - Groenelaan, fietspad	41.93
1.1A - Waterranonkellaan 23	44.2
…	…

Table 12. Example of output file Data_result.csv used for historic data

Annex D: Training Data Preparation

```
import csv
import os
from datetime import datetime

# Local file path to save the result substitute “Date_fixed_data_path” with
your own
result_directory = "C:/Date_fixed_data_path/"
if not os.path.exists(result_directory):
    os.makedirs(result_directory)

# Prompt the user to enter the path of the CSV file substitute
“file_to_read_path” with the path leading to the file with the sensor names
csv_file_path = "C:/file_to_read_path/GWT_SENSOR_MEAN_VALUES.csv"
column_number = 2
count = 0

with open(csv_file_path, 'r', encoding='utf-8') as file:
    reader = csv.reader(file)
    # Skip the header row
    next(reader)

    # Loop through the remaining rows
    for row in reader:
        # Extract the first word from the first column
        original_value = row[0]

        file_path = row[0].split()[0]

      # Local file path to read the CSV file, substitute
“path_to_save_new_sensorfiles”
      with your own path.
        file_path0 = f"C:/path_to_save_new_sensorfiles/{file_path}.csv"

        with open(file_path0, 'r', encoding='utf-8') as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=';')

            # Skip the lines until reaching the "datum" column
            for _ in range(14):
                try:
                    next(csv_reader)
                except StopIteration:
                    # If there are no more rows, continue to the next file
                    continue

            # Initialize a list to store the converted rows
            converted_rows = []
            # Loop through the remaining rows and convert the values
            for row in csv_reader:
                datum = datetime.strptime(row[0], "%d-%m-%Y").strftime("%d-%m-
%Y")
                tijd = datetime.strptime(row[1], "%H:%M").strftime("%I:%M:%S
%p")
                meetwaarde = float(row[2]) / 100
                converted_rows.append([f"{datum}{tijd}", meetwaarde])

        # Create the result file
        result_file = f"{result_directory}{file_path}.csv"
        with open(result_file, 'w', newline='') as result:
            writer = csv.writer(result)
            writer.writerow(["Date_Time", "Depth(m)"])  # Add the header
            writer.writerows(converted_rows)

        count += 1
        print(f"Converted file {file_path}.csv and saved as {result_file}...
{count} files so far...")

print("Processing of all CSV files completed.")


# Check and fix date format for all result files
total_files = sum(1 for filename in os.listdir(result_directory) if
filename.endswith(".csv"))
fixed_files = 0
for filename in os.listdir(result_directory):
    if filename.endswith(".csv"):
        file_path = os.path.join(result_directory, filename)
        updated_rows = []
        with open(file_path, 'r', newline='') as result:
            reader = csv.reader(result)
            try:
                headers = next(reader)  # Skip the headers
            except StopIteration:
                # Skip empty result file
                continue

            for row in reader:
                date_time = row[0]
                depth = row[1]
                try:
                    date_time_fixed = datetime.strptime(date_time, "%d/%m/%Y
%I:%M:%S %p").strftime("%d-%m-%Y %I:%M:%S %p")
                except ValueError:
                    try:
                        date_time_fixed = datetime.strptime(date_time,
"%d/%m/%Y %H:%M:%S %p").strftime("%d-%m-%Y %I:%M:%S %p")
                    except ValueError:
                        try:
                            date_time_fixed = datetime.strptime(date_time,
"%d/%m/%Y %I:%M %p").strftime("%d-%m-%Y %I:%M:%S %p")
                        except ValueError:
                            try:
                                date_time_fixed = datetime.strptime(date_time,
"%d/%m/%Y %H:%M").strftime("%d-%m-%Y %I:%M:%S %p")
                            except ValueError:
                                # Keep the original value if it cannot be
parsed
                                date_time_fixed = date_time

                updated_rows.append([date_time_fixed, depth])

        with open(file_path, 'w', newline='') as result:
            writer = csv.writer(result)
            writer.writerow(headers)
            writer.writerows(updated_rows)

        fixed_files += 1
        print(f"Checking and fixing date format in {filename}... {fixed_files}
out of {total_files} files fixed so far...")

print("Checking and fixing date format in result files completed.")
```

The script processes the source CSV files (downloaded with the Annex B: Data Collection Code (Python), converts and saves the data into new result files, and then checks and fixes the date-time format in these result files as some of the older registries are not consistent with the way the date data was stored. These data has to be as homogenous as possible as the machine learning algorithm will attempt to find patterns associated with the date and GWT depths.

1. Directory Handling and Prompt:
   - The script sets up the directory path where the result files will be saved.
   - It checks if the directory exists and creates it if not.

2. CSV Processing Loop (First Part):
   - The script reads the source CSV file using `csv.reader`.
   - It skips the header row using `next(reader)`.

3. Loop Through Source CSV Rows:
   - The script loops through the rows of the source CSV file.
   - For each row, it extracts the first word from the first column.
   - It constructs a file path based on the extracted value.

4. Convert and Save Data to Result Files:
- The script reads each CSV file specified by the constructed file path.
- It skips rows until it reaches the relevant column.
- It converts the data values (date, time, and depth) and stores them in a list of converted rows.

5. Create and Write Result Files:
- The script constructs the path for the result file using the directory and the file path.
- It creates the result file and writes the header and converted rows to it.

6. Completion Message (First Part):
- The script counts the processed files and prints a status message.

7. Date Format Checking and Fixing Loop:
- The script counts the total number of CSV files in the result directory.
- It loops through each result file in the directory.

8. Read and Update Result Files:
- For each result file, the script opens it using `csv.reader`.
- It skips the headers and reads each row.
- It attempts to parse and fix the datetime format using multiple formats, falling back to the original if parsing fails.
- It updates the rows with fixed datetime format.

9. Write Updated Result Files:
- The script opens each result file for writing and writes the updated rows.

10. Completion Message (Second Part):
- The script counts the fixed files and prints a status message until it is finished.

Date_Time	Depth(m)
04-06-197412:00:00 PM	32.04
18-06-197412:00:00 PM	32.1
04-07-197412:00:00 PM	32.18
18-07-197412:00:00 PM	32.23
04-08-197412:00:00 PM	32.33
…	…

Table 13. Example of date format fixed on file 011A.

Annex E: Join GWT Data with Rainfall Data (RH in mm)

```
import csv
import os
from datetime import datetime

# File paths substitute "file_to_read_path", "joined_data_output" and
"Date_fixed_data_path" with your own
csv_directory = "C:/Date_fixed_data_path"
csv_file_path = "C:/file_to_read_path/GWT_SENSOR_MEAN_VALUES.csv"
output_file_path = "C:/joined_data_output/Rainfall_training_updated.csv"

# Read the Date_Time_fix values from Rainfall_training_updated.csv
date_time_fix_values = {}
with open(output_file_path, 'r', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        date_time_fix_values[row['Date_Time_fix']] = row['RH(mm)']

# Process each CSV file in the directory
for filename in os.listdir(csv_directory):
    if filename.endswith(".csv"):
        file_path = os.path.join(csv_directory, filename)

        # Skip the GWT_SENSOR_MEAN_VALUES.csv file
        if file_path == csv_file_path:
            continue

        # Read the rows from the current CSV file
        rows = []
        with open(file_path, 'r', newline='') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                rows.append(row)

        # Proceed only if rows are available
        if rows:
            # Check if 'RH(mm)' field is already present in the fieldnames
            fieldnames = list(rows[0].keys())
            if 'RH(mm)' not in fieldnames:
                fieldnames.append('RH(mm)')

            # Update the rows with matching RH(mm) values
            for row in rows:
                # Check if 'Date_Time' key exists in the row
                if 'Date_Time' in row:
                    # Convert the Date_Time value to match the format in
Date_Time_fix
```

```
                    date_time = datetime.strptime(row['Date_Time'], '%d-%m-
%Y%I:%M:%S %p')
                    date_time = date_time.strftime('%d-%m-%Y%I:%M:%S %p')

                    # Retrieve the RH(mm) value from the dictionary
                    rh_mm = date_time_fix_values.get(date_time)

                    if rh_mm is not None:
                        # Update the RH(mm) value in the current row
                        row['RH(mm)'] = rh_mm

            # Write the updated rows back to the CSV file
            with open(file_path, 'w', newline='') as csvfile:
                writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
                writer.writeheader()
                writer.writerows(rows)

            print(f"Processed file: {file_path}")
        else:
            print(f"Skipped empty file: {file_path}")

print("All files processed successfully.")
```

This code reads and processes CSV files in a specified directory. It reads `Date_Time_fix` and `RH(mm)` values from a CSV file that contains the RH (Rainfall) values for the dates from 1987 to the latest updated day (this data was obtained from the Twente Airport weather station from KNMI) , and then iterates through the prepared training data CSV files in the same directory, updating their `'RH(mm)'` values based on the corresponding `Date_Time` values found in the previously read CSV file. The processed rows are then written back to the CSV files. If there are any blank values in the data used this might generate some "#Value!" values that will not be recognised by other subsequent codes, therefore that should be considered while preparing the data.

1. Import Statements:
- The necessary libraries are imported: `csv`, `os`, and `datetime`.

2. File Paths:
- Various file paths are defined, including the directory containing CSV files (`csv_directory`), the path to the main CSV file (`csv_file_path`), and the path to the output CSV file (`output_file_path`).

3. Read Date_Time_fix Values:
- The code reads the `Rainfall_training_updated.csv` file to extract the `Date_Time_fix` and corresponding `RH(mm)` values. These values are stored in a dictionary named `date_time_fix_values`.

4. Process Each CSV File:
    - The code iterates through each file in the `csv_directory`.

5. Skip GWT_SENSOR_MEAN_VALUES.csv File:
    - It checks if the current file path is the same as the `csv_file_path` to skip processing the main CSV file.

6. Read Rows from CSV:
    - The rows from the current CSV file are read and stored in the `rows` list.

7. Check If Rows Are Available:
    - The code checks if there are any rows available in the current CSV file.

8. Update Fieldnames and Rows:
    - If rows are available, it checks if the fieldname `'RH(mm)'` is present in the fieldnames of the CSV.
    - If not present, it appends `'RH(mm)'` to the fieldnames.
    - It then iterates through each row and processes them.

9. Processing Each Row:
    - For each row, it first checks if `'Date_Time'` exists in the row.
    - If present, it converts the `'Date_Time'` value to match the format in `'Date_Time_fix'` and retrieves the corresponding `'RH(mm)'` value from the `date_time_fix_values` dictionary.

10. Update RH(mm) Value:
    - If a `'RH(mm)'` value is found in the dictionary, it updates the `'RH(mm)'` value in the current row.

11. Write Updated Rows to CSV:
    - After processing all rows, it writes the updated rows back to the same CSV file.
    - It uses the `csv.DictWriter` to write the rows, including the updated `'RH(mm)'` value.

12. Printing and Status Updates:
    - The code prints a message indicating the processing of each file. If a file is empty (no rows), it is skipped and a relevant message is printed.

13. Completion Message:
    - After processing all files, a completion message is printed.

Date	HH	RH	RH(mm)	Date_Time_fix
19740601	12:00:00 AM	0	0	01-06-197412:00:00 AM
19740601	1:00:00 AM	-1	0.05	01-06-197401:00:00 AM
19740601	2:00:00 AM	-1	0.05	01-06-197402:00:00 AM
19740601	3:00:00 AM	-1	0.05	01-06-197403:00:00 AM
19740601	4:00:00 AM	-1	0.05	01-06-197404:00:00 AM
…	…	…	…	…

Table 14. Example of the file Rainfall_training_updated.csv

Date_Time	Depth(m)	RH(mm)
04-06-197412:00:00 PM	32.04	0
18-06-197412:00:00 PM	32.1	0
04-07-197412:00:00 PM	32.18	0.4
18-07-197412:00:00 PM	32.23	0.05
04-08-197412:00:00 PM	32.33	0
…	…	…

Table 15. Result example of joined 011A with RH(mm) values from Rainfall_training_update.csv

Annex F: Real-time GWT Mesh Generator Code – Button A (C# - Unity)

```
using System.Collections;
using System.IO;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(MeshFilter))]
public class ScriptA : MonoBehaviour
{

    Mesh mesh; //how we will call meshes

    Vector3[] vertices;
    int[] triangles;

    public string filePath; // The path to the CSV file in the Assets folder
    public int columnIndex; // The index of the column you want to read (0-
based)
    public float updateInterval = 5f; // The interval in seconds to update the
value
    public waterpump waterPumpScript; // Reference to the waterpump script


    private float lastUpdateTime; // The time of the last update
    private float value; // The value read from the CSV file
    private float value1; // The value read from the CSV file
    private float value2; // The value read from the CSV file
    private float value3; // The value read from the CSV file
    private float value4; // The value read from the CSV file
    private float value5;
    private float value6;
    private float value7;
    private float value8;
    private float value9;
    private float value10;
    private float value11;
    private float value12;
    private float value13;
    private float value14;
    private float value15;
    private float value16;
    private float value17;
    private float value18;
    private float value19;
    private float value20;
    private float value21;
    private float value22;
    private float value23;
    private float value24;
    private float value25;
    private float value26;
    private float value27;
    private float value28;
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
    private float value273;

    private List<GameObject> spheres; // List to store the sphere objects

    int start = 0;

    int s = 0;
    GameObject previousBackMesh;


    public bool isActive = false;

    void Update()
    {



        if (isActive)
        {
            Debug.Log("THIS IS CODE A RUNNING."); // Log message
                                                  // Implement behaviour for
ScriptA when active
                                                  // Call the restart method
of the other script


            if (start==0)
            {
                // Initialize the spheres list
                spheres = new List<GameObject>();

                // Call the UpdateValue function for the first time
                UpdateValue();

                CreateShape();

                CreateShape2();

                UpdateMesh();

                start = 1;
            }


            // Check if it's time to update the value
            if (Time.time - lastUpdateTime >= updateInterval)
            {

                UpdateValue();
                UpdateMesh();
                Destroy(previousBackMesh);
                s = 0;
                CreateShape();
                CreateShape2();

            }
        }

    }

    void UpdateValue()
    {
        // Read the contents of the CSV file
        string[] lines = File.ReadAllLines(filePath);

        // Initialize an array to store the values
        float[] values = new float[lines.Length];

        // Loop over each row and extract the value from the second column
        for (int i = 0; i < lines.Length; i++)
        {
            string[] columns = lines[i].Split(',');
            float.TryParse(columns[1], out values[i]);
        }

        // Assign the values to the variables
        value = values[0];
        value1 = values[1];
        value2 = values[2];
        value3 = values[3];
        value4 = values[4];
        value5 = values[5];
        value6 = values[6];
        value7 = values[7];
        value8 = values[8];
        value9 = values[9];
        value10 = values[10];
```

```
        value11 = values[11];
        value12 = values[12];
        value13 = values[13];
        value14 = values[14];
        value15 = values[15];
        value16 = values[16];
        value17 = values[17];
        value18 = values[18];
        value19 = values[19];
        value20 = values[20];
        value21 = values[21];
        value22 = values[22];
        value23 = values[23];
        value24 = values[24];
        value25 = values[25];
        value26 = values[26];
        value27 = values[27];
        value28 = values[28];
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
        value273 = values[273];



        // Update the last update time
        lastUpdateTime = Time.time;

        // Debugging output
        //Debug.Log("Value: " + value + ", " + value1 + ", " + value2 + ", " +
value3 + ", " + value4);
    }


    void CreateShape2()
    {
        // ...

        float sphereHeight = 1f; // Set the desired height for all spheres

        List<int> excludedVertices = new List<int>() { 3, 4, 8, 25, 52, 54,
66, 109, 120, 128, 133, 154, 238, 262 }; // List of excluded vertex indices

        // Destroy previously created spheres
        GameObject[] previousSpheres =
GameObject.FindGameObjectsWithTag("PlayerArmature");
        foreach (GameObject sphere in previousSpheres)
        {
            Destroy(sphere);
        }



        for (int i = 0; i < vertices.Length; i++)
        {
            if (excludedVertices.Contains(i))
            {
                continue; // Skip creating a sphere for excluded vertices
            }

            Vector3 vertexPosition = vertices[i];
            float sphereYPosition = vertexPosition.y - (sphereHeight / 2f); //
Adjust the position so the top part aligns with the vertex height

            // Instantiate a sphere at the adjusted position with the fixed
height
            GameObject sphere =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
            sphere.transform.position = new Vector3(vertexPosition.x,
sphereYPosition, vertexPosition.z);
            sphere.transform.localScale = new Vector3(1f, sphereHeight, 1f);
// Scale the sphere in the y-direction
            sphere.tag = "PlayerArmature";

            Rigidbody rb = sphere.AddComponent<Rigidbody>();
            rb.useGravity = false;

            Collider col = sphere.GetComponent<Collider>();
            col.isTrigger = true;

            // Add the sphere to the list
            spheres.Add(sphere);

            // Get the sphere's radius
            float sphereRadius = sphere.transform.localScale.x * 0.5f;

            // Set the collider size to match the sphere's size
            if (col is SphereCollider sphereCollider)
            {
                sphereCollider.radius = sphereRadius;
            }
            else if (col is CapsuleCollider capsuleCollider)
            {
                // If using a capsule collider, adjust the size accordingly
                capsuleCollider.radius = sphereRadius;
                capsuleCollider.height = sphereRadius * 2f;
            }
        }

        // ...
    }



    private void OnDrawGizmosSelected()
    {
        if (spheres != null)
        {
            Gizmos.color = Color.red;
            foreach (var sphere in spheres)
            {
                if (sphere != null)
                {
                    SphereCollider sphereCollider =
sphere.GetComponent<SphereCollider>();
                    if (sphereCollider != null)
                    {
                        Gizmos.DrawWireSphere(sphere.transform.position,
sphereCollider.radius);
                    }
                }
            }
        }
    }

    void CreateShape()
    {
        //float distance = -1000.0f; // The distance to extend the shape
downwards



        vertices = new Vector3[]
        {
        new Vector3((float)1279.42603, value, (float)-2725.45728), //0
        new Vector3((float)-1359.4494, value1, (float)-3646.86646), //1
        new Vector3((float)876.575928, value2, (float)-1428.83582), //2
        new Vector3((float)0, value3, (float)0), //3
        new Vector3((float)0, value4, (float)0), //4
        new Vector3((float)3046.35156, value5, (float)-1563.30457), //5
        new Vector3((float)-656.8895, value6, (float)3855.567), //6
        new Vector3((float)-3532, value7,(float)452), //7
        new Vector3((float)0, value8, (float)0), //8
        new Vector3((float)2021.85156, value9, (float)55.8551254), //9
        new Vector3((float)2248.81592, value10, (float)571.831909), //10
        new Vector3((float)2143.01855, value11, (float)939.67981), //11
        new Vector3((float)323.509155, value12, (float)-1478.48962), //12
        new Vector3((float)577.314697, value13, (float)-1206.94226), //13
        new Vector3((float)612, value14, (float)-1405.93665), //14
        new Vector3((float)1305.13135, value15, (float)398.776276), //15
        new Vector3((float)-5323.5693, value16, (float)-9.39585114), //16
        new Vector3((float)-5445.6088, value17, (float)1257.09509), //17
        new Vector3((float)-5793.4511, value18, (float)685.586914), //18
        new Vector3((float)2406.86377, value19, (float)2777.59424), //19
        new Vector3((float)-583.48071, value20, (float)850.385376), //20
        new Vector3((float)-519.62097, value21, (float)1187.77698), //21
        new Vector3((float)2012.48926, value22, (float)-11.803688), //22
        new Vector3((float)-5549.7377, value23, (float)714.868408), //23
        new Vector3((float)-3541.2700, value24, (float)-584.823486), //24
        new Vector3((float)0, value25, (float)0), //25
        new Vector3((float)-4141.6601, value26, (float)-458.712921), //26
        new Vector3((float)-4362.6044, value27, (float)-412.567413), //27
        new Vector3((float)-5505.8964, value28, (float)570.140503), //28
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
        new Vector3((float)-1507.5228, value273, (float)-1678.48792) //273


        //ATTENTION!** last one should not have a comma at the end***
        };





        triangles = new int[]
        {
            212, 205, 134,
            134, 213, 212,
            132, 159, 134,
            159, 6, 134,
            204, 132, 134,
            19, 211, 212,
            205, 204, 134,
            213, 19, 212,
            211, 205, 212,
            206, 204, 205,
            211, 206, 205,
            70, 213, 81,
            70, 19, 213,
            51, 6, 159,
            19, 210, 211,
            214, 19, 70,
            204, 159, 132,
            206, 203, 204,
            207, 203, 206,
            210, 206, 211,
            214, 210, 19,
            210, 207, 206,
            203, 159, 204,
            203, 201, 159,
            194, 6, 51,
            159, 200, 51,
            196, 194, 51,
            201, 200, 159,
            200, 196, 51,
            210, 208, 207,
            207, 202, 203,
            214, 209, 210,
//The code had to be cut from the to avoid it being too long to share
            226, 1, 0,
            96, 186, 184

        };


    }

    // Update is called once per frame
    void UpdateMesh()
    {



        Mesh frontMesh = new Mesh();
        frontMesh.Clear();
        frontMesh.vertices = vertices;
        frontMesh.triangles = triangles;
        frontMesh.RecalculateNormals();
        GetComponent<MeshFilter>().mesh = frontMesh;

        Mesh backMesh = new Mesh();
        backMesh.Clear();
        backMesh.vertices = vertices;
        int[] backTriangles = new int[triangles.Length];
        for (int j = 0; j < triangles.Length; j++)
        {
            backTriangles[j] = triangles[triangles.Length - j - 1];
        }
        backMesh.triangles = backTriangles;
        backMesh.RecalculateNormals();

        GameObject backMeshObject = new GameObject("BackMesh" + s);
        previousBackMesh = GameObject.Find("BackMesh" + s);
        s = s + 1;


        backMeshObject.transform.parent = transform;
        backMeshObject.transform.localPosition = Vector3.zero;
        backMeshObject.AddComponent<MeshFilter>().mesh = backMesh;
        backMeshObject.AddComponent<MeshRenderer>().material =
GetComponent<MeshRenderer>().material;


    }


    // Call this method when Button A is pressed
    public void OnButtonAPressed()
    {
        if (waterPumpScript != null)
        {
            waterPumpScript.DestroyCylinder(); // Call the DestroyCylinder
method in the waterpump script
        }
    }

}
```

The script reads data from the provided CSV file (The real-time data generated by the Python code), generates a 3D mesh based on that data, and creates spheres at specific vertices (that have collision boxes that will be used to identify when the GWT touches or is at a certain range from the surface). These is done while updating the mesh periodically. The final result is a 3D shape with dynamically updated values and interactive elements in the Unity environment.

This C# script represents a Unity MonoBehaviour script that generates a 3D mesh based on data read from a CSV file. Let's break down the important components of the code:

1. Imports and Component Declaration: The script includes the necessary Unity and system libraries and declares the class `GWT_277`, which extends `MonoBehaviour` and requires a `MeshFilter` component.

2. Variables Declaration: Various variables are declared to store mesh data, file paths, and other necessary information. Notably, the `vertices` and `triangles` arrays hold the 3D vertex positions and triangle indices to form the mesh. Other variables like `filePath`, `updateInterval`, and `spheres` will be used for CSV file path, update interval for mesh update, and storing sphere game objects, respectively.

3. Start() Method: This method is called once when the script is initialised. It initialises the `spheres` list, calls `UpdateValue()`, `CreateShape()`, `CreateShape2()`, and `UpdateMesh()` functions.

4. Update() Method: The `Update()` method is called every frame. It checks if the specified time interval (`updateInterval`) has passed since the last update. If so, it calls `UpdateValue()`, `UpdateMesh()`, `Destroy()` the previous back mesh, resets `s` to 0, and then calls `CreateShape()` and `CreateShape2()`.

5. UpdateValue() Method: This method reads the CSV file data from `filePath` and extracts specific values from the second column of each row. The extracted values are stored in various float variables (e.g., `value`, `value1`, ..., `value28`, `value273`). The `Debug.Log()` function outputs the extracted values for debugging purposes.

6. CreateShape() and CreateShape2() Methods: These methods create a 3D shape by defining the `vertices` array and the `triangles` array. The `CreateShape()` method initialises `vertices` with specific 3D coordinates, while `CreateShape2()` modifies `vertices` by excluding certain vertices based on the `excludedVertices` list and instantiates spheres at the remaining vertices.

7. UpdateMesh() Method: This method updates the mesh using the `vertices` and `triangles` arrays. It creates a new `Mesh` object for the front mesh and back mesh, calculates normals, and applies the front mesh to the current game object. Additionally, it creates a game object for the back mesh, which is a mirrored version of the front mesh, and positions it at the origin.

011A - Wethouder Horstman Sportpark	32.002
017A - Rokkerkamp 2A	45.84132
075A - 39 Mediant	34.418
075B - 39 Mediant	34.358
075C - 39 Mediant	34.325
080B - Perseusstraat 147	29.336
083A - Groenelaan, fietspad	42.912
1.1A - Waterranonkellaan 23	44.0294
1.1B - Waterranonkellaan 23	43.9873
1001 - Poolmansweg 187	29.2636
…	…

Table 16. Example of resultant real-time updated file named Consolidated_V1.csv used in Unity

Annex G: Historic Data Code - Button B and C (C# Unity)

```
using System.Collections;
using System.IO;
using System.Collections.Generic;
using UnityEngine;


[RequireComponent(typeof(MeshFilter))]
public class ScriptB : MonoBehaviour
{

    Mesh mesh; //how we will call meshes

    Vector3[] vertices;
    int[] triangles;

    public string filePath2; // The path to the CSV file in the Assets folder
    public int columnIndex2; // The index of the column you want to read (0-
based)
    public float updateInterval2 = 10f; // The interval in seconds to update
the value
    public float amount = 0.001f;

    public float lastUpdateTime; // The time of the last update
    public float value; // The value read from the CSV file
    public float value1; // The value read from the CSV file
    public float value2; // The value read from the CSV file
    public float value3; // The value read from the CSV file
    public float value4; // The value read from the CSV file
    public float value5;
    public float value6;
    public float value7;
    public float value8;
    public float value9;
    public float value10;
    public float value11;
    public float value12;
    public float value13;
    public float value14;
    public float value15;
    public float value16;
    public float value17;
    public float value18;
    public float value19;
    public float value20;
    public float value21;
    public float value22;
    public float value23;
    public float value24;
    public float value25;
    public float value26;
    public float value27;
    public float value28;
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
    public float value273;

    private List<GameObject> spheres; // List to store the sphere objects

    int start = 0;
    int s = 0;
    GameObject previousBackMesh;

    public bool isActive = false;

    void Update()
    {
        if (isActive)
        {
            Debug.Log("THIS IS CODE B RUNNING."); // Log message
                                                  // Implement behaviour for
ScriptB when active

            if (start == 0)
            {
                // Initialize the spheres list
                spheres = new List<GameObject>();

                // Call the UpdateValue function for the first time
                UpdateValue();

                CreateShape();

                CreateShape2();

                UpdateMesh();

                start = 1;
            }


            // Check if it's time to update the value
            if (Time.time - lastUpdateTime >= updateInterval2)
            {
                Decrease();
                UpdateMesh();
                Destroy(previousBackMesh);
                s = 0;
                CreateShape();
                CreateShape2();

            }

        }
    }

    public void RestartScript()
    {
        // Restart the script
        UpdateValue();
    }



    void Decrease()
    {
        value -= amount;
        value1 -= amount;
        value2 -= amount;
        value3 -= amount;
        value4 -= amount;
        value5 -= amount;
        value6 -= amount;
        value7 -= amount;
        value8 -= amount;
        value9 -= amount;
        value10 -= amount;
        value11 -= amount;
        value12 -= amount;
        value13 -= amount;
        value14 -= amount;
        value15 -= amount;
        value16 -= amount;
        value17 -= amount;
        value18 -= amount;
        value19 -= amount;
        value20 -= amount;
        value21 -= amount;
        value22 -= amount;
        value23 -= amount;
        value24 -= amount;
        value25 -= amount;
        value26 -= amount;
        value27 -= amount;
        //The code had to be cut from the 28 value and skipped to the 273 to 
avoid it being too long to share
        value273 -= amount;


    }
    void UpdateValue()
    {
        // Read the contents of the CSV file
        string[] lines = File.ReadAllLines(filePath2);

        // Initialize an array to store the values
        float[] values = new float[lines.Length];

        // Loop over each row and extract the value from the second column
        for (int i = 0; i < lines.Length; i++)
        {
            string[] columns = lines[i].Split(',');
            float.TryParse(columns[1], out values[i]);
        }

        // Assign the values to the variables
        value = values[0];
        value1 = values[1];
        value2 = values[2];
        value3 = values[3];
        value4 = values[4];
        value5 = values[5];
        value6 = values[6];
        value7 = values[7];
        value8 = values[8];
        value9 = values[9];
        value10 = values[10];
        value11 = values[11];
        value12 = values[12];
        value13 = values[13];
        value14 = values[14];
        value15 = values[15];
        value16 = values[16];
        value17 = values[17];
        value18 = values[18];
        value19 = values[19];
        value20 = values[20];
        value21 = values[21];
        value22 = values[22];
        value23 = values[23];
        value24 = values[24];
        value25 = values[25];
        value26 = values[26];
        value27 = values[27];
        value28 = values[28];
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
        value273 = values[273];
        // Update the last update time
        lastUpdateTime = Time.time;
    }


    void CreateShape2()
    {
        // ...

        float sphereHeight = 1f; // Set the desired height for all spheres

        List<int> excludedVertices = new List<int>() { 3, 4, 8, 25, 52, 54,
66, 109, 120, 128, 133, 154, 238, 262 }; // List of excluded vertex indices

        // Destroy previously created spheres
        GameObject[] previousSpheres =
GameObject.FindGameObjectsWithTag("PlayerArmature");
        foreach (GameObject sphere in previousSpheres)
        {
            Destroy(sphere);
        }




        for (int i = 0; i < vertices.Length; i++)
        {
            if (excludedVertices.Contains(i))
            {
                continue; // Skip creating a sphere for excluded vertices
            }

            Vector3 vertexPosition = vertices[i];
            float sphereYPosition = vertexPosition.y - (sphereHeight / 2f); //
Adjust the position so the top part aligns with the vertex height

            // Instantiate a sphere at the adjusted position with the fixed
height
            GameObject sphere =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
```

```
            sphere.transform.position = new Vector3(vertexPosition.x,
sphereYPosition, vertexPosition.z);
            sphere.transform.localScale = new Vector3(1f, sphereHeight, 1f);
// Scale the sphere in the y-direction
            sphere.tag = "PlayerArmature";

            Rigidbody rb = sphere.AddComponent<Rigidbody>();
            rb.useGravity = false;

            Collider col = sphere.GetComponent<Collider>();
            col.isTrigger = true;

            // Add the sphere to the list
            spheres.Add(sphere);

            // Get the sphere's radius
            float sphereRadius = sphere.transform.localScale.x * 0.5f;

            // Set the collider size to match the sphere's size
            if (col is SphereCollider sphereCollider)
            {
                sphereCollider.radius = sphereRadius;
            }
            else if (col is CapsuleCollider capsuleCollider)
            {
                // If using a capsule collider, adjust the size accordingly
                capsuleCollider.radius = sphereRadius;
                capsuleCollider.height = sphereRadius * 2f;
            }
        }

        // ...
    }



    void CreateShape()
    {
        //float distance = -1000.0f; // The distance to extend the shape
downwards



        vertices = new Vector3[]
        {
        new Vector3((float)1279.42603, value, (float)-2725.45728), //0
        new Vector3((float)-1359.4494, value1, (float)-3646.86646), //1
        new Vector3((float)876.575928, value2, (float)-1428.83582), //2
        new Vector3((float)0, value3, (float)0), //3
        new Vector3((float)0, value4, (float)0), //4
        new Vector3((float)3046.35156, value5, (float)-1563.30457), //5
        new Vector3((float)-656.8895, value6, (float)3855.567), //6
        new Vector3((float)-3532, value7,(float)452), //7
        new Vector3((float)0, value8, (float)0), //8
        new Vector3((float)2021.85156, value9, (float)55.8551254), //9
        new Vector3((float)2248.81592, value10, (float)571.831909), //10
        new Vector3((float)2143.01855, value11, (float)939.67981), //11
        new Vector3((float)323.509155, value12, (float)-1478.48962), //12
        new Vector3((float)577.314697, value13, (float)-1206.94226), //13
        new Vector3((float)612, value14, (float)-1405.93665), //14
        new Vector3((float)1305.13135, value15, (float)398.776276), //15
        new Vector3((float)-5323.5693, value16, (float)-9.39585114), //16
        new Vector3((float)-5445.6088, value17, (float)1257.09509), //17
        new Vector3((float)-5793.4511, value18, (float)685.586914), //18
        new Vector3((float)2406.86377, value19, (float)2777.59424), //19
        new Vector3((float)-583.48071, value20, (float)850.385376), //20
        new Vector3((float)-519.62097, value21, (float)1187.77698), //21
        new Vector3((float)2012.48926, value22, (float)-11.803688), //22
        new Vector3((float)-5549.7377, value23, (float)714.868408), //23
        new Vector3((float)-3541.2700, value24, (float)-584.823486), //24
        new Vector3((float)0, value25, (float)0), //25
        new Vector3((float)-4141.6601, value26, (float)-458.712921), //26
        new Vector3((float)-4362.6044, value27, (float)-412.567413), //27
        new Vector3((float)-5505.8964, value28, (float)570.140503), //28
//The code had to be cut from the 28 value and skipped to the 273 to avoid it
being too long to share
        new Vector3((float)-1507.5228, value273, (float)-1678.48792) //273


        //ATTENTION!** last one should not have a comma at the end***
        };






        triangles = new int[]
        {
            212, 205, 134,
            134, 213, 212,
            132, 159, 134,
            159, 6, 134,
            204, 132, 134,
            19, 211, 212,
            205, 204, 134,
            213, 19, 212,
            211, 205, 212,
            206, 204, 205,
            211, 206, 205,
            70, 213, 81,
            70, 19, 213,
            51, 6, 159,
            19, 210, 211,
            214, 19, 70,
            204, 159, 132,
            206, 203, 204,
            207, 203, 206,
            210, 206, 211,
            214, 210, 19,
            210, 207, 206,
            203, 159, 204,
            203, 201, 159,
            194, 6, 51,
            159, 200, 51,
            196, 194, 51,
            201, 200, 159,
            200, 196, 51,
            210, 208, 207,
            207, 202, 203,
            214, 209, 210,
            202, 201, 203,
            209, 208, 210,
            40, 207, 208,
            40, 202, 207,
            195, 194, 196,
            200, 63, 62,
            201, 63, 200,
            201, 179, 31,
            202, 179, 201,
            217, 209, 214,
            31, 63, 201,
            199, 196, 200,
            62, 199, 200,
            193, 6, 194,
            70, 216, 214,
            40, 30, 202,
            81, 216, 70,
            199, 195, 196,
            216, 217, 214,
            172, 208, 209,
            208, 141, 40,
            172, 141, 208,
            0, 1, 87,
            226, 1, 0,
            96, 186, 184

        };

    }

    // Update is called once per frame
    void UpdateMesh()
    {
        Mesh frontMesh = new Mesh();
        frontMesh.Clear();
        frontMesh.vertices = vertices;
        frontMesh.triangles = triangles;
        frontMesh.RecalculateNormals();
        GetComponent<MeshFilter>().mesh = frontMesh;

        Mesh backMesh = new Mesh();
        backMesh.Clear();
        backMesh.vertices = vertices;
        int[] backTriangles = new int[triangles.Length];
        for (int j = 0; j < triangles.Length; j++)
        {
            backTriangles[j] = triangles[triangles.Length - j - 1];
        }
        backMesh.triangles = backTriangles;
        backMesh.RecalculateNormals();

        GameObject backMeshObject = new GameObject("BackMesh" + s);
        previousBackMesh = GameObject.Find("BackMesh" + s);
        s = s + 1;


        backMeshObject.transform.parent = transform;
        backMeshObject.transform.localPosition = Vector3.zero;
        backMeshObject.AddComponent<MeshFilter>().mesh = backMesh;
        backMeshObject.AddComponent<MeshRenderer>().material =
GetComponent<MeshRenderer>().material;
    }
}
```

This C# script manages the creation and manipulation of a 3D mesh for the representation of the GWT. It reads data from the Historic CSV file to update values that influence the mesh's shape, then dynamically adjusts the mesh over time, creating spheres at certain vertices to visualise the changes. this script manipulates mesh shapes in a Unity scene. It reads values from a CSV file to drive changes in the mesh's appearance, animating it over time. The script demonstrates the dynamic modification of mesh geometry to create interactive visual effects in a game or interactive application. The file from Table 12. Example of output file Data_result.csv is used in conjunction with this script to load the historic values.

1. Start():
    - Initialises the spheres list to store GameObjects.
    - Calls the UpdateValue() function to read initial values from a CSV file.
    - Calls CreateShape() and CreateShape2() functions to create mesh shapes.
    - Calls UpdateMesh() to update the mesh's appearance based on shapes.
2. Update():
    - Checks if it's time for an update based on the defined updateInterval2.
    - If an update is due, calls Decrease() to decrease stored values.
    - Calls UpdateMesh() to update the mesh's appearance based on new values.
    - Destroys the previous back mesh to prepare for the next update.
    - Calls CreateShape() and CreateShape2() to recreate the mesh shapes.
3. Decrease():
    - Decreases the values of the GWT by the specified amount. This is used to animate the simulation of changes in the mesh.
4. UpdateValue():
    - Reads values from a CSV file.
    - Assigns the read values to multiple variables (value, value1, ..., value273).
    - Updates the lastUpdateTime to the current time.
5. CreateShape():
    - Defines the vertices and triangles that form the main mesh shape.
    - Updates the mesh's vertices and triangles based on the defined shape.
6. CreateShape2():
    - Creates additional shapes (spheres) based on specific vertices.
    - Instantiates sphere GameObjects, positions them based on vertices, and scales them.
    - Assigns Rigidbody and Collider components to the spheres for physics interactions.
    - Adds the spheres to the spheres list.
7. UpdateMesh():
    - Updates the main mesh's appearance using the calculated vertices and triangles.
    - Creates a back mesh (mirror) with reversed triangles to simulate a 3D effect.
    - Destroys the previous back mesh to prevent accumulation of game objects.

Annex H: Surface Collision Code (C# -Unity)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;


public class waterpump : MonoBehaviour
{
    // Declare a reference to the cylinder GameObject
    GameObject cylinder;
    GameObject pumpDeciCopy; // Declare the pumpDeciCopy variable


    // Reference to the animated object to be copied
    public GameObject pumpDeciPrefab;
    public float yOffset = 20f;
    private bool pump = true;

    void OnTriggerEnter(Collider other)
    {
        // Check if the collision was with a specific object
        if (other != null && other.gameObject.CompareTag("PlayerArmature"))
        {
            {
            // Perform an action when the collision occurs
            //Debug.Log("Object collided ENTERED with PlayerArmature");


            // Change the color of the collided object to red
            Renderer renderer = other.gameObject.GetComponent<Renderer>();
            if (renderer != null)
            {
                renderer.material.color = Color.red;
            }



            // Destroy the previously created cylinder (if exists)
            if (cylinder != null)
            {
                Destroy(cylinder);

            }

            // Destroy the previously created pumpDeciCopy (if exists)
            if (pumpDeciCopy != null)
            {
                Destroy(pumpDeciCopy);
                    StartCoroutine(DelayedCommand());
                }

            // Create a tall cylinder GameObject
            cylinder = GameObject.CreatePrimitive(PrimitiveType.Cylinder);
            cylinder.transform.position = other.transform.position;
            cylinder.transform.localScale = new Vector3(10f, 1000f, 10f); //
Adjust the height of the cylinder as desired

            // Add a semi-transparent material to the cylinder
            Material cylinderMaterial = new
Material(Shader.Find("Transparent/Diffuse"));
            cylinderMaterial.color = new Color(1f, 0f, 0f, 0.5f); // Set the
color with transparency
            cylinder.GetComponent<Renderer>().material = cylinderMaterial;

            // Instantiate a copy of the animated object at the collision
position with an offset
             Vector3 pumpDeciPosition = other.transform.position + new
Vector3(0f, yOffset, 0f);
             pumpDeciCopy = Instantiate(pumpDeciPrefab, pumpDeciPosition,
Quaternion.identity);

            }






            // Send a command to IFTTT using HTTP request
            StartCoroutine(SendIFTTTCommandON());
        }

    }


    private IEnumerator DelayedCommand()
    {
        // Perform some actions before the delay

        //Debug.Log("Before delay");

        yield return new WaitForSeconds(15.0f); // Add a delay of 2 seconds

        StartCoroutine(SendIFTTTCommandOFF());

        // Perform some actions after the delay
        //Debug.Log("After delay");
    }


    void OnTriggerExit(Collider other)
    {
        // Check if the collision was with a specific object
        if (other.gameObject.CompareTag("PlayerArmature"))
        {
            // Perform an action when the collision occurs
            Debug.Log("Object collided EXITED with PlayerArmature");

            // Remove the color and revert back to the original material
            Renderer renderer = other.gameObject.GetComponent<Renderer>();
            if (renderer != null)
            {
                renderer.material.color = renderer.sharedMaterial.color;
            }



            // Destroy the previously created cylinder (if exists)
            if (cylinder != null)
            {
                Destroy(cylinder);

            }

            // Destroy the previously created pumpDeciCopy (if exists)
            if (pumpDeciCopy != null)
            {
                Destroy(pumpDeciCopy);
            }

            // Send a command to IFTTT using HTTP request
            StartCoroutine(SendIFTTTCommandOFF());
        }
    }


    IEnumerator SendIFTTTCommandON()
    {

        if (pump==true)
        {
            // Construct the HTTP request URL with your IFTTT key and event
name
```

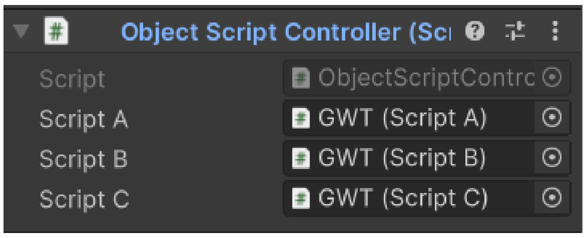
```
            string url = 
"https://maker.ifttt.com/trigger/WaterpumpON/json/with/key/ji0z0qFEjjZ-
Psisk9b6_7xRfufAoxpsS";

            // Send the HTTP request
            UnityWebRequest www = UnityWebRequest.Get(url);
            yield return www.SendWebRequest();

            // Check if there was an error
            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError("IFTTT command failed: " + www.error);
            }
            else
            {
                Debug.Log("IFTTT ON command sent successfully");
                pump = false;
            }

        }
    }

    // Destroy the cylinder GameObject
    public void DestroyCylinder()
    {
        if (cylinder != null)
        {
            Destroy(cylinder);
            Destroy(pumpDeciCopy);
        }
    }

    IEnumerator SendIFTTTCommandOFF()
    {
        // Construct the HTTP request URL with your IFTTT key and event name
        string url = 
"https://maker.ifttt.com/trigger/WaterpumpOFF/json/with/key/ji0z0qFEjjZ-
Psisk9b6_7xRfufAoxpsS";

        // Send the HTTP request
        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        // Check if there was an error
        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("IFTTT command failed: " + www.error);
        }
        else
        {
            Debug.Log("IFTTT OFF command sent successfully");
        }
    }
}
```

This script creates interactive behaviour in a Unity scene. When a collider with a specific tag enters the trigger zone of the GameObject, it changes the colour of the collided object, creates a cylinder and an animated object copy of the water pump to be shown in the point of collision and allow the user to know exactly where the water has entered the range to activate the water pump. Then, it sends a command to IFTTT for the "ON" state. When the collider exits the trigger zone, it reverts the colour, destroys the cylinder and the animated object copy, and sends a command to IFTTT for the "OFF" state. The script includes coroutines to introduce delays and send HTTP requests to IFTTT, enabling dynamic interactions based on collider triggers. It is important to note that the code will not work without a proper IFTTT key, this key can be obtained by creating an account on ifttt.com.

1. OnTriggerEnter(Collider other):
    - This function is called when a Collider enters the trigger zone of the GameObject this script is attached to (usually referred to as a "trigger" in Unity).
    - Checks if the entering collider's GameObject has a specific tag ("PlayerArmature").
    - If the tag matches, the following actions are performed:
        - Changes the colour of the collided object to red.
        - Destroys any previously created cylinder.
        - Destroys any previously created pumpDeciCopy and then starts a coroutine DelayedCommand() after a delay.
        - Creates a tall cylinder GameObject with transparency, representing the pump's "on" state.
        - Instantiates a copy of the animated object (pumpDeciPrefab) at the collision position with an offset.
        - Sends a command to IFTTT using an HTTP request by calling SendIFTTTCommandON() coroutine.
2. DelayedCommand():
    - This coroutine introduces a delay before performing actions.
    - The coroutine pauses for a specified amount of time (15 seconds in this case) using yield return new WaitForSeconds().
    - After the delay, it starts another coroutine SendIFTTTCommandOFF() to send a command to IFTTT.
3. OnTriggerExit(Collider other):
    - This function is called when a Collider exits the trigger zone.
    - Checks if the exiting collider's GameObject has a specific tag ("PlayerArmature").
    - If the tag matches, it performs the following actions:
        - Logs a message to indicate the collision exit.
        - Reverts the colour of the collided object to its original material.
        - Destroys any previously created cylinder.
        - Destroys any previously created pumpDeciCopy.

        - Sends a command to IFTTT using an HTTP request by calling SendIFTTTCommandOFF() coroutine.
4. SendIFTTTCommandON():
    - This coroutine sends an "ON" command to IFTTT using an HTTP request.
    - It checks if the pump is already in an "ON" state (controlled by the pump variable).
    - If the pump is in the "ON" state, it constructs an HTTP request URL with the IFTTT key and event name.
    - Sends the HTTP request using Unity's UnityWebRequest and checks for success or failure.
    - If successful, logs a message indicating the successful command and updates the pump state to false.
5. SendIFTTTCommandOFF():
    - This coroutine sends an "OFF" command to IFTTT using an HTTP request.
    - It constructs an HTTP request URL with the IFTTT key and event name.
    - Sends the HTTP request using Unity's UnityWebRequest and checks for success or failure.
    - If successful, logs a message indicating the successful command.
6. Variables and Declarations:
    - Declare variables to reference the cylinder GameObject and the animated object (pumpDeciCopy).
    - Publicly expose a GameObject reference (pumpDeciPrefab) and a float variable (yOffset) to control the offset of the instantiated object.
    - Declare a boolean variable pump to control the state of the pump ("ON" or "OFF").

Annex I: Button Control (Buttons A, B, C) Code (C# -Unity)



Script	ObjectScriptContrc
Script A	GWT (Script A)
Script B	GWT (Script B)
Script C	GWT (Script C)

Figure 28. Assigning Script A and Script B to be accessed by the Historic_switch script.

```
using UnityEngine;
using UnityEngine.UI;

public class ObjectScriptController : MonoBehaviour
{
    public ScriptA scriptA;
    public ScriptB scriptB;
    public ScriptC scriptC;

    public void ActivateScriptA()
    {
        scriptA.isActive = true;
        scriptB.isActive = false;
        scriptC.isActive = false;

        scriptB.RestartScript();
    }

    public void ActivateScriptB()
    {
        scriptA.isActive = false;
        scriptB.isActive = true;
        scriptC.isActive = false;
    }

    public void ActivateScriptC()
    {
        scriptA.isActive = false;
        scriptB.isActive = false;
        scriptC.isActive = true;

        scriptB.RestartScript();
    }
}
```

This script allows the user to switch between two other scripts when the button is pressed ('scriptA: That later is assigned as the one that loads generates the GWT mesh in real-time' and `scriptB: That later is assigned as the script that loads Historical Data`) and controlling their behaviour. It toggles between real-time and historical data views, updates UI elements by changing the name of the button to the next possible selection, and sends an HTTP request to an IFTTT service when switching to the real-time view.

1. Using Statements:
   - `using System.Collections;`: This is used for working with collections and enumerators.
   - `using System.Collections.Generic;`: This is used for working with generic collections.
   - `using UnityEngine;`: This provides access to Unity's scripting APIs.
   - `using UnityEngine.UI;`: This provides access to Unity's UI-related components.
   - `using UnityEngine.Networking;`: This is used for network-related operations.

2. Class Declaration:
   - `public class Historic_switch : MonoBehaviour`: This declares a public class named `Historic_switch` that inherits from `MonoBehaviour`, the base class for Unity scripts.

3. Public Variables:
   - `public GWT_277 scriptA;` and `public GWT_277_HISTORICAL scriptB;`: These are public references to other scripts (`GWT_277` and `GWT_277_HISTORICAL`).
   - `public Text buttonText;`: This is a public reference to a UI Text component.

4. Private Variables:
   - `private bool showButton = false;`: This is a private boolean variable used to control whether a button should be shown.
   - `private bool scriptAIsActive = true;`: This is a private boolean variable that keeps track of whether `scriptA` is active.

5. Start Method:
   - `public void Start()`: This is a public method that Unity automatically calls when the script starts.

   Inside this method:
   - The `scriptA` and `scriptB` scripts are enabled/disabled based on the value of `scriptAIsActive`.
   - The `showButton` variable is set to `false`.

6. SwitchScript Method:
   - `public void SwitchScript()`: This is a public method that is triggered when a button associated with this script is clicked.

   Inside this method:
   - The value of `scriptAIsActive` is toggled.

   Based on `scriptAIsActive`:
   - Either `scriptA` is enabled and `scriptB` is disabled, or vice versa.
   - The `buttonText` is updated to indicate whether the active data is realtime or historical.
   - The `RestartScript` method of `scriptB` is called.
   - The `SendIFTTTCommandOFF` coroutine is started (sends a command to IFTTT).

7. SendIFTTTCommandOFF Coroutine:
   - `IEnumerator SendIFTTTCommandOFF()`: This is a coroutine method for sending an HTTP request to an IFTTT service.

   Inside this coroutine:
   - The URL for the IFTTT service is constructed.
   - An HTTP request is created using `UnityWebRequest.Get(url)`.
   - The request is sent using `yield return www.SendWebRequest()`.
   - The result of the request is checked for success or error, and appropriate debug messages are logged.

Annex J: Camera Toggle Code (C# -Unity)

This script allows to toggle between different cameras in the scene using the switchcam(int x) function. It stores the initial positions and rotations of all cameras in arrays when the script starts. The deactivateall() function turns off all cameras, and the switchcam(int x) function resets the positions and rotations of cameras and enables the desired camera based on the provided integer parameter x.
This setup could is useful for implementing camera switching in a scene, such as moving between different viewpoints or perspectives, in the case of the model a Top, Side and Isometric preestablished views were implemented. In order for it to work, this script must be applied to the canvas holding the buttons, an then selecting the specific cameras that will be implemented with the buttons. Therefore each button is assigned an "On Click ()" event were the desired camera is selected.

1. Variables Declaration:
    - Three public Camera variables (cam1, cam2, cam3) represent the different cameras in the scene.
    - Private arrays startingPositions and startingRotations are used to store the initial positions and rotations of all cameras.
2. Start():
    - This method is executed when the script is started.
    - It finds all the cameras in the scene using FindObjectsOfType<Camera>().
    - Then, it saves the initial positions and rotations of each camera into the startingPositions and startingRotations arrays.
3. switchcam(int x):
    - This function is used to switch between different cameras in the scene based on the integer parameter x.
    - It first calls the deactivateall() function to turn off all cameras.
    - Depending on the value of x, it performs the following:
        - If x is 1, it resets the position and rotation of all cameras to their initial values, then enables cam1.
        - If x is 2, it resets the position and rotation of all cameras to their initial values, then enables cam2.
        - Otherwise, it resets the position and rotation of all cameras to their initial values, then enables cam3.
4. deactivateall():
    - This function is used to disable all cameras by turning off their enabled property.

Annex K: Camera Controls Code (C# -Unity)

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

/// <summary>
/// A simple free camera to be added to a Unity game object.
///
/// Keys:
/// wasd / arrows   - movement
/// q/e             - up/down (local space)
/// r/f             - up/down (world space)
/// pageup/pagedown - up/down (world space)
/// hold shift      - enable fast movement mode
/// right mouse     - enable free look
/// mouse           - free look / rotation
///
/// </summary>
public class camera_move : MonoBehaviour
{
    /// <summary>
    /// Normal speed of camera movement.
    /// </summary>
    public float movementSpeed = 50f;

    /// <summary>
    /// Speed of camera movement when shift is held down,
    /// </summary>
    public float fastMovementSpeed = 500f;

    /// <summary>
    /// Sensitivity for free look.
    /// </summary>
    public float freeLookSensitivity = 4f;

    /// <summary>
    /// Amount to zoom the camera when using the mouse wheel.
    /// </summary>
    public float zoomSensitivity = 50f;

    /// <summary>
    /// Amount to zoom the camera when using the mouse wheel (fast mode).
    /// </summary>
    public float fastZoomSensitivity = 500f;
    /// <summary>
    /// Set to true when free looking (on right mouse button).
    /// </summary>
    private bool looking = false;

    void Update()
    {
        var fastMode = Input.GetKey(KeyCode.LeftShift) ||
Input.GetKey(KeyCode.RightShift);
        var movementSpeed = fastMode ? this.fastMovementSpeed :
this.movementSpeed;

        if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
        {
            transform.position = transform.position + (-transform.right *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
        {
            transform.position = transform.position + (transform.right *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
        {
            transform.position = transform.position + (transform.forward *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow))
        {
            transform.position = transform.position + (-transform.forward *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.Q))
        {
            transform.position = transform.position + (transform.up *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.E))
        {
            transform.position = transform.position + (-transform.up *
movementSpeed * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.R) || Input.GetKey(KeyCode.PageUp))
        {
            transform.position = transform.position + (Vector3.up *
movementSpeed * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.F) || Input.GetKey(KeyCode.PageDown))
        {
            transform.position = transform.position + (-Vector3.up *
movementSpeed * Time.deltaTime);
        }

        if (looking)
        {
            float newRotationX = transform.localEulerAngles.y +
Input.GetAxis("Mouse X") * freeLookSensitivity;
            float newRotationY = transform.localEulerAngles.x -
Input.GetAxis("Mouse Y") * freeLookSensitivity;
            transform.localEulerAngles = new Vector3(newRotationY,
newRotationX, 0f);
        }

        float axis = Input.GetAxis("Mouse ScrollWheel");
        if (axis != 0)
        {
            var zoomSensitivity = fastMode ? this.fastZoomSensitivity :
this.zoomSensitivity;
            transform.position = transform.position + transform.forward * axis
* zoomSensitivity;
        }

        if (Input.GetKeyDown(KeyCode.Mouse1))
        {
            StartLooking();
        }
        else if (Input.GetKeyUp(KeyCode.Mouse1))
        {
            StopLooking();
        }
    }

    void OnDisable()
    {
        StopLooking();
    }

    /// <summary>
    /// Enable free looking.
    /// </summary>
    public void StartLooking()
    {
        looking = true;
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
    }

    /// <summary>
    /// Disable free looking.
    /// </summary>
    public void StopLooking()
    {
        looking = false;
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }
}
```

This script was used directly from (Davis 2019) it creates a simple free camera movement and rotation system. Users can control the camera's movement using the WASD keys, arrow keys, and specific keys for up and down movement. The camera can also be rotated freely using the right mouse button while in free look mode. Mouse scroll wheel controls zooming. The script provides both normal and fast modes for movement and zooming. In order fot it to work, this script must be added to each individual camera that is desired to be controlled.

1. Variables Declaration:
    - Various public float variables define movement and sensitivity parameters for the camera.
    - A private boolean variable looking is used to track whether the camera is in free look mode.
2. Update():
    - This function is called every frame to handle user input and update the camera's position and orientation.
    - It determines whether the camera is in fast mode (Shift key held down) and sets the appropriate movement and zoom speeds.
    - Based on user input, it adjusts the camera's position using the transform.position property. For example, Input.GetKey(KeyCode.A) moves the camera to the left.
3. Looking with Mouse:
    - If the looking flag is set (right mouse button is held down), the script adjusts the camera's local Euler angles (transform.localEulerAngles) based on the mouse movement. This creates a free look effect where the camera rotates around its pivot.
4. Zoom with Mouse Scroll Wheel:
    - The script checks the mouse scroll wheel input (Input.GetAxis("Mouse ScrollWheel")) to zoom the camera in and out. The zoom sensitivity depends on the camera's current mode (fast or normal).
5. Mouse Right Click:
    - If the right mouse button is pressed (Input.GetKeyDown(KeyCode.Mouse1)), the StartLooking() function is called. This function sets up free look mode by hiding the

    cursor, locking it in the center of the screen, and enabling camera rotation with mouse movement.
  - When the right mouse button is released (Input.GetKeyUp(KeyCode.Mouse1)), the StopLooking() function is called to disable free look mode.
6. StartLooking() and StopLooking():
   - These functions control the free look mode. StartLooking() enables free look by setting looking to true and locking the cursor, while StopLooking() disables free look and restores the cursor's visibility and behaviour.
7. OnDisable():
   - This function is triggered when the script is disabled. It ensures that free look mode is deactivated by calling StopLooking().

Annex L: GWT Decision Tree Predictive Model Code (Google Colab)

```
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import os


# Directory containing CSV files
csv_directory = "/content/"
csv_mean_values_file = "/content/GWT_SENSOR_MEAN_VALUES.csv"

# Read the mean values file to get the order of the files
mean_values_data = pd.read_csv(csv_mean_values_file)
file_order = mean_values_data['name'].apply(lambda name: name.split()[0])  #
Use the first word of the name column

# Get a list of all .csv files in the directory
csv_files = [file for file in os.listdir(csv_directory) if
file.endswith('.csv')]

# Initialize an empty DataFrame to store results
result_df = pd.DataFrame(columns=['File_Name', 'Highest_Predicted_Depth'])

# Get user input for RH(mm) values per hour
input_rh = []
for hour in range(24):
    rh = float(input(f"Enter RH(mm) for {hour:02d}:00 - "))
    input_rh.append(rh)

# Iterate through each CSV file in the order from the mean values file
for csv_file in file_order:
    try:
        # Read the data from the current CSV file
        data = pd.read_csv(os.path.join(csv_directory, f"{csv_file}.csv"))

        if 'Date_Time' not in data.columns:
            print(f"Warning: 'Date_Time' column not found in {csv_file}.csv.
Skipping...")
            continue

        # Convert Date_Time to numerical representation
        data['Date_Time'] = pd.to_datetime(data['Date_Time'], format='%d-%m-
%Y%H:%M:%S %p')
        data['Date_Num'] = data['Date_Time'].dt.strftime('%y%m%d').astype(int)
        data['Hour_Num'] = data['Date_Time'].dt.strftime('%H%M').astype(int)
        # Convert columns to numeric, handling non-numeric values as NaN
        numeric_columns = ['Depth(m)', 'Date_Num', 'Hour_Num', 'RH(mm)']
        numeric_columns_present = [col for col in numeric_columns if col in
data.columns]
        data[numeric_columns_present] =
data[numeric_columns_present].apply(pd.to_numeric, errors='coerce')

        # Impute missing values
        imputer = SimpleImputer(strategy='mean')
        data[numeric_columns_present] =
imputer.fit_transform(data[numeric_columns_present])

        # Prepare the input features (RH(mm)) and target variable (Depth(m))
        X = data[['Date_Num', 'Hour_Num', 'RH(mm)']].values
        y = data['Depth(m)'].values.astype(float)  # Convert 'Depth(m)' to
float

        # Train the model
        model = DecisionTreeRegressor()
        model.fit(X, y)

        # Get the current date and time
        current_date = datetime.now().strftime('%y%m%d')

        # Generate the input data for prediction
        input_data = np.column_stack((np.repeat(int(current_date), 24),
np.arange(0, 2400, 100), input_rh))

        # Predict the Depth(m) for the next day
        predicted_depths = model.predict(input_data)

        # Find and save the highest predicted depth
        highest_predicted_depth = max(predicted_depths)
        result_df = result_df.append({'File_Name': f"{csv_file}.csv",
'Highest_Predicted_Depth': highest_predicted_depth}, ignore_index=True)

    except ValueError as e:
        print(f"Error processing {csv_file}.csv: {e}")
        # Handle empty file case
        result_df = result_df.append({'File_Name': f"{csv_file}.csv",
'Highest_Predicted_Depth': np.nan}, ignore_index=True)

# Save the results to a consolidated CSV file
consolidated_result_file = "/content/consolidated_results.csv"
result_df.to_csv(consolidated_result_file, index=False)
```

```
print("Prediction and consolidation completed. Results saved to:",
consolidated_result_file)
```

This code involves working with data from KNMI API, data preprocessing with Pandas, machine learning model training and evaluation, and writing prediction results to a CSV file. It requires the file GWT_SENSOR_MEAN_VALUES.csv and the sensor individual files to be uploaded in the "/content/" folder in Google Colab.

1. Installing Required Package:
    - The script starts by installing the `knmy` package using `!pip install knmy`. This will allow to access the KNMI weather stations data.

2. Import Statements:
    - The necessary libraries are imported, including datetime handling, data manipulation with Pandas, machine learning tools from scikitlearn, and CSV handling.

3. Get File Names from CSV:
    - The script reads a CSV file named `GWT_SENSOR_MEAN_VALUES.csv` to get a list of file names. It skips the header row and extracts the first word from each row.

4. Create and Prepare Result File:
    - A result file named `PREDICTIONS.csv` is created to store the prediction results.
    - The header ("File Name", "Max Depth") is written to the result file.

5. Loop Through File Names:
    - For each file name obtained, the script proceeds to process the corresponding data.

6. Data Retrieval from KNMI API:
    - The script prompts the user to enter a date for prediction.
    - It fetches RH (Hourly Rainfall) values for a specified date using the KNMI API.

7. Data Preprocessing:
    - RH values are converted to a list and normalised.
    - Another CSV file is read, and Date_Time columns are converted to numerical formats.
    - Input features (`X`) and target variables (`y`) are prepared.´
    - The data contains the Date, GWT Depth and the Rainfall (in mm) therefore when the

8. Model Training:
    - The dataset is split into training and testing sets.
    - A Decision Tree Regressor model is trained on the training set.

9. Model Evaluation:
    - The model is evaluated on the testing set using mean squared error.

10. Generate Input Data for Prediction:
    - The script generates input data for prediction based on the specified date and RH values.

11. Predict Depth for Specified Date:
    - The script predicts the Depth(m) values for the specified date using the trained model.

12. Print Predicted Depths:
    - The predicted Depths for each hour of the specified date are printed.

The data prepared in Table 15. Result example of joined 011A with RH(mm) values from Rainfall_training_update.csv per sensor is used to train the model and later used with the RH(mm) predicted values from a weather site like Buienradar to generate predictions. The resultant file is called consolidated_results.csv and this file is used to load the highest predicted values for the next day per sensor in Unity.

File_Name	Highest_Predicted_Depth
011A.csv	31.02
017A.csv	45.84862
075A.csv	34.418
075B.csv	32.69
075C.csv	32.55
080B.csv	29.336
083A.csv	42.4501
1.1A.csv	44.3756
1.1B.csv	44.3615
…	…

Table 17. Example of results from the prediction code on the file consolidated_results.csv

Annex M: Ethical Considerations, Risks, and Contingencies

Any research project must take ethical considerations, risks, and contingencies into account. The following ethical issues, dangers, and contingencies have been noted in this project:

1. Data Privacy and Security: Care was taken to make sure that the project only uses open data from reputable sources. Due to the fact that the data is already available to the public, there is less chance of personal information from the study area being leaked by the research. Before publishing the finished products, any collected data that may compromise the privacy of people, companies, or organizations was appropriately anonymized or cleaned up.

2. Data's shareability: The project relies on data obtained from KNMI and Twents Waternet, which was carefully evaluated for its shareability. In cases where the obtained data was not possible to be shared, "dummy" data was generated with the same format and structure. This allowed to still understand the research for training purposes and provide readers with a clear understanding of the framework until they are able to acquire their own data.

3. Impact on House Pricing: The geolocated warnings and alerts produced by the monitoring system may have an effect on the pricing of particular homes. Therefore, the following actions need to be considered:

   a. Any proposed action plan derived from this research should prioritize the safety of citizens over any political or economic interests.

   b. Any early alerts derived from the use of this framework should be internally managed by the organization and as possible involve the municipality of the study area to be discussed with relevant stakeholders in order to work on preventive solutions, avoiding causing unnecessary alarm among neighbourhoods or communities.

   c. Warnings should only be issued publicly when risks, such as flooding, are beyond the control of the municipality and stakeholders. Efforts then should be focused on evacuation or preparation to prevent tragedies and economic losses.

   d. It is recommended that any outcomes obtained through this framework and intended for public dissemination should be shared within the neighbourhood block scope. This practice aims to prevent the identification of individual households.

   e. When conducting public regional tests of the monitoring framework, it is important to promptly inform the community to avoid spreading incorrect information, causing unnecessary alarm, or desensitizing them to valid alerts.

These ethical considerations, along with the identified risks and corresponding contingencies, demonstrate a commitment to conducting the research in a responsible and ethical manner, prioritizing data privacy, community safety, and accurate information dissemination.

Annex N: First Framework Version Questionnaire

Questions	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
A. User interface					
The system's terms are easily understandable.	0	1	2	3	4
Navigating through different parts of the system is easy.	0	1	2	3	4
Error messages are easily understandable.	0	1	2	3	4
The delay between operations is acceptable.	0	1	2	3	4
It's easy to return to the homepage.	0	1	2	3	4
B. Spatial interface					
Moving to a new location on the map is easy.	0	1	2	3	4
Zooming in and out on the map is easy.	0	1	2	3	4
Creating new content is easy.	0	1	2	3	4
Accessing information about what is displayed on the map is easy.	0	1	2	3	4
Visual edits on the map take effect immediately.	0	1	2	3	4
C. Learnability					
Confidence in using the system.	0	1	2	3	4
Remembering how to perform tasks is easy.	0	1	2	3	4
Discovering new features by trial and error is easy.	0	1	2	3	4
Finding the help resources useful.	0	1	2	3	4
Easily undoing mistakes.	0	1	2	3	4
D. Effectiveness					
The system provides tools to reach my goals.	0	1	2	3	4
The system is reliable.	0	1	2	3	4
Completing tasks that would be impossible without the system.	0	1	2	3	4
Increased participation in the project due to the system.	0	1	2	3	4
Recommendation of the system to others.	0	1	2	3	4
E. Communication					
Helping communicate ideas to other participants.	0	1	2	3	4
Always understanding what the system is showing.	0	1	2	3	4
Easy understanding of the maps.	0	1	2	3	4
Ability to express opinions about other participants' ideas.	0	1	2	3	4
Having assistance available when having a problem.	0	1	2	3	4

Table 18. Questions to valuate initial impression for the first version of the GWT working framework based on (Aguilar 2022).