3D House Simulation - Virtual Reality in Training and Education

Ronan Hanley

Dylan Loftus

B.Sc. (Hons) in Software Development

May 6, 2020

Final Year Project

Advised by: Damien Costello
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Contents

About this project

Abstract In schools and workplaces today, education and training is provided on a teacher to student basis. Teachers provide the information, and the student or employee memorizes that information. This is an effective way of learning, but is there a better way?

As Virtual Reality becomes more and more accessible, companies jump on the idea of a more cost effective, risk free way to train their new employees. Universities and Colleges have also started using Virtual Training as a learning tool.

But does this simulated way of training provide the same results that we would see in the real world using more conventional types of learning (books, videos, training days)? In our Final Year Project, we explore this idea by creating a 3D environment containing a house, along with simulated temperatures based on real world weather data. Using this real world data, we can simulate how temperature flows from the outside of the house to the inside. We can also simulate how temperature flows from one room to another inside the house. We use this simulation as an example of how an individual can learn in a virtual environment.

Authors The authors are Ronan Hanley and Dylan Loftus, both graduating in 2020 with a B.Sc (Honours) in Computing in Software Development from Galway-Mayo Institute of Technology.

» Link to GitHub Repository «

Chapter 1

Introduction

1.1 Work Allocation

Under the wiki section of our GitHub Repository we have two Effort Log Pages. It was recommended by DXC that we create an effort log each, which we added to for each day that we worked on the project. We wrote down the date, what work we planned to carry out, the problems that we encountered, and how we fixed those problems. The effort logs are a good way of seeing in detail what work we both carried out.

1.1.1 Unity and the Web Service

- Initial research: **Ronan and Dylan**
- Initial Unity scene (rooms, player movement), initial JSONReader implementation: **Dylan**
- Doors and windows (sprite and functionality): **Dylan**
- Weather API (OpenAPI spec, Flask server, weather data processing, ...): **Ronan**
- Temperature representation through floor colour: **Ronan and Dylan**
- Temperature equalisation code: **Ronan**
- Temperature display within rooms: **Dylan**
- Radiators (model and functionality): **Ronan**
- Minimap: Started by **Ronan**, completed by **Dylan**

- Oculus implementation: **Dylan**
- Modular scene (+ modular room models and hallway prefabs): **Dylan**
- Room spawning in modular scene: **Dylan**
- Dynamic temperature equalisation for modular scene: **Ronan**
- Gamification elements (comfort and money): **Ronan**
- Top left UI (weather timestamp, comfort, money, progress slider): **Ronan**
- Weather select scene: **Ronan**
- Day night cycle: **Dylan**
- Skybox and terrain: **Dylan**
- Lighting and lightmaps: **Dylan and Ronan**

1.1.2 Sections of the Dissertation

- Abstract: **Dylan**
- Introduction: **Dylan**
- Methodology: **Dylan**
- Technology Review: **Ronan**
- System Design: **Ronan and Dylan**
- System Evaluation: **Ronan**
- Conclusion: **Ronan and Dylan**

This project had a main objective of investigating the use of Virtual Reality as an interactive learning resource. For this project, the context was that of energy and heat transfer between rooms.

A brief description of each chapter in this dissertation:

- **Introduction** : The introduction covers the themes explored in this project, the requirements of the project, the objectives of the project, and also gives a brief explanation of each chapter within this dissertation.

- **Methodology** : The methodology covers how we went about the development of the project, and the software methodologies we used throughout the project. We talk about what platforms we decided to use for the project, and we also outline how we went about meeting with our project supervisor and DXC.
- **Technology Review** : The technology review discusses each aspect of the project (both front-end and back-end), discussing each component in great detail.
- **System Design** : The system design explains the architecture of the project. It also presents a UML diagram of the project architecture.
- **System Evaluation** : The system evaluation evaluates the final product. Is the software robust? Did we produce answers to all of the questions that we asked in the introduction? Furthermore, what opportunities does the project provide?
- **Conclusion** : The conclusion clearly states the answers to our main research question, and then summarises and reflects on that research. It also includes our final thoughts on the project.

1.2 Theme of the Project

Virtual Reality is a way of placing a person into a computer-generated simulation or environment that can be interacted with. It is a great way to immerse a person in an environment that can almost be as realistic as a real world one. A virtual environment can be created by replacing real-world elements with computer-generated ones [1]. These computer-generated environments create endless opportunities for any software developer. Scenarios range from placing the user at the center of a derelict city in a post apocalyptic setting for video game entertainment purposes, or generating a simulation that replicates a real life scenario for training or learning purposes. For example, a simulation taking place inside of a retail store that teaches the user how to operate a till. This project aims to find out if Virtual Reality is a good medium for learning or training. Does this simulated way of training provide the same results, if not better results, than what we would see in the real world?

1.3 Objective of the Project

In the real world we have no way of physically seeing temperature flow, we just feel the effects of it. The main concept of this project is to demonstrate the effectiveness of using a virtual reality headset as a medium for learning. In order to do this we place the user into a virtual environment simulation that accurately shows the flow of temperature in a typical housing setting.

1.4 Video Games

A video game is a game played by at least one user, in which the user electronically manipulates images produced by a computer program on a monitor or other display, for example a television. From one of the very first video games Spacewar! in 1962 to today, the gaming industry makes more money than the movie and music industries combined [2]. There are over 1 billion people in the world that play video games. There are also a vast array of video gaming genres, for example action, role-playing, sports, and many more. Today it's commonplace that some people spend hours in their day playing these video games. If we could somehow incorporate this level of engagement and motivation when it comes to learning/training, who knows what kind of results could be produced? With the amount of money that the industry is making and the huge amount of people that play video games, it is safe to say that video games are not something to be overlooked.

1.5 Simulations

A simulation is a computer program that represents something. It could be anything from flying a plane to using a till in a restaurant.

Advantages and Disadvantages of Simulations in Training

In 1998 a study [3] was conducted that sought to find the advantages and disadvantages of simulations when they were used for testing and training with scenarios from complex problem-solving. These simulations were PC-based.

Advantages

- Fast Results: The paper found that "PC-based simulations allow for a quick computing of results".

- The opportunity for practice: The paper found that the users can practice and experiment their ideas with a complex system, and in turn by doing this, learn how the system works without risk or cost. “They can actively discover and explore an unknown scenario and acquire declarative as well as procedural knowledge about a certain domain of reality” [3].
- Motivation: The paper found that the users thought the “PC-based simulations provoked challenge and curiosity”. This promotes engagement.
- Feedback: The users were able to get feedback quickly and could adapt accordingly to provide better results.
- Adaptability: The simulation could be changed very easily depending on the training objectives and the user’s needs.

Disadvantages

- Not knowing the scale: PC-based simulations are often so complex that even the developer of the system does not know what the best way to sort the particular problem presented in the simulation. If the developer doesn’t know what the best way to perform the task is, then this poses quite the problem for comparing individual results. If there is no “best-way” to solve a particular problem then results are scored relative to each other.
- Comparing results: The paper found that the results from a PC-based simulation cannot be easily compared from one user to another. In a “free simulation”, an action performed by one user can drastically change their simulation environment, making comparisons difficult or impossible.

1.6 Game/Simulation based Learning/Training

“A motivated learner can never be stopped.” [4] Game based learning/training is a way of playing or having fun while at the same time retaining information that can be applied in the real world. Video games can hold a user’s attention for hour after hour, day after day, whilst if you tried to teach them about thermodynamics through a video or in person, it’s safe to say you’ll find it hard to keep their attention at maximum level. Game based learning combines the content of a particular subject, may it be maths or science, with the motivation and engagement that video games brings to the table. [4]

Advantages and Disadvantages of Game Based Learning/Training

In a 2009 paper [5] advantages and disadvantages were given by a group of students in relation to "browser-game-based learning in mathematics education".

Advantages

- Improved mathematical skills: One of the students stated that the sum of the score that they received in the game was an example of their improved mathematical skills.
- Learn with ease: Most of the students stated that the games made learning easy.
- More understandable: With the use of 3D objects, the students told that these games made the topic "more concrete and understandable". [5]

Although game based learning can provide positive results and has advantages, it also comes with some disadvantages found in the 2009 paper [5].

Disadvantages

- Eye defects: Some of the students stated that the games could possibly lead to eye defects but they also added that the learning benefits may outweigh this disadvantage.
- Possible better alternatives: One of the students said that they found these types of games useless and stated that playing chess would benefit a person more when it comes to mathematics.

1.7 Project Implementation

3D House Simulator is a project idea that was provided to us from DXC technology located in Ballybrit Business Park in Galway. "3D House Simulator" is a simulation replicating a single floor home with configurable rooms, where the temperature can be regulated in each room by adjusting the heating controls. The user can navigate around the house, add rooms and hallways, open and close doors or windows, and turn radiators on or off. The temperature values are taken from real world weather stations, and the user has the option before the simulation starts to select what time of year the simulation will take place. Within the simulation the main goal that the user has is to regulate the temperature of each room to keep their comfort percentage at an adequate level. As the temperature outside drops, the rooms will get colder meaning that the user will have to leave the radiators on for a longer period of time. Similarly if the room is too hot (and it is cold outside) the user can open a window to lower the temperature within the room. Depending on what temperature an adjacent room is, the user could also open the current room's door to allow temperatures to equalize between connecting rooms.

The temperature of each room is visible on the mini-map and is also visualised by the colour of the floor. The colour shifts on a gradient, with blue colours indicating a cold room, and red colours indicating a hot room.

Chapter 2

Methodology

At the start of our project we were given the opportunity to take on a project provided to us by DXC technology. We assessed the challenges and requirements put towards us and we found them interesting, challenging, and fun to take on.

2.1 Meetings

Initially we had weekly in-house meetings with our project supervisor Damien in the college and Skype meetings with Indy, Ed, Maria, and Shivam from DXC. In these introductory meetings we would discuss possible architecture approaches and set out a list of goals which we hoped to achieve by the end of the project. After two weeks of having these meetings, we found it best to have bi-weekly meetings with DXC and continue having weekly meetings with Damien.

From then on we went with an iterative approach in regards to the development of the simulation, each taking a goal and trying to complete that goal in time for the next meeting.

2.2 Platforms

2.2.1 Head Mounted Display

One of the requirements for this project was to have the project be deployable to a Virtual Reality Headset, or HMD (Head Mounted Display) for short.

2.2.2 Google Cardboard

Initially we didn't have direct access to one of these headsets, so we scouted the internet looking for a potential alternative for our project until we gained access to one of the better headsets, for example the HTC Vive or Oculus Rift. During our research, our project supervisor Damien informed us that he was in possession of a few Google Cardboards. This was perfect for us to get the ball rolling in VR development in Unity. To use the Google cardboard with our Unity Project we had to make sure that we had Android Build Support enabled during our installation of Unity. After that, we downloaded and imported the Google VR SDK (software development kit) into our Unity Project. This gave us a few sample scenes with all the necessary assets imported and placed on GameObjects to enable the functionality of the Google Cardboard. We ported these assets over to our project, creating a new player GameObject that used the Google VR functionality. After all this was done, an android version of the project had to be built and mounted to a mobile phone. The phone was then placed into the Google cardboard and the simulation could then be viewed in 360 degrees of virtual space. This was a very limited approach though, as there was only one way of getting input from the user: a touch on the screen. We wanted to give the user many more inputs than this (which would be available when we got our hands on a better headset) but it was good for now as it taught us how to import SDK's and utilise them.

2.2.3 Oculus Quest

A few times during the development of the project, we would have in house meetings with DXC. At our first in-house meeting, we demonstrated an early build of our project along with the Google Cardboard build of the project. We then discussed about getting the project deployed to a HMD. We were then brought to a room in the DXC offices where they showed us an Oculus Quest and let us play around with it for a few minutes. We played a demo of Space Pirate Trainer where you fight off relentless waves of droids. The overall experience was surprisingly immersive. After using the headset we discussed that it was probably best if we used the same headset for the development of our project, as if we ran into any problems, DXC would be there to help us. We got in contact with our project supervisor Damien and informed him on what happened in our meeting with DXC. We then all came to the conclusion that the Oculus Quest would be used as a platform for our project. A few weeks later we had access to an Oculus Quest and began developing a build of our project that was compatible with the headset.

The development path we took to get a build of our project on the Oculus Quest was very similar to the approach we took with the Google Cardboard. The Oculus Quest also ran on Android, so that was very fortunate for us, as we didn't have to install Unity again with any extra components. To start off we had to download the Oculus Integration Package, and we could to this through the Unity Asset Store within the Unity editor itself. This provided us with a few sample scenes, prefabs, and scripts with all the necessary components for Oculus Quest functionality. We then created a player object in our scene in our project and attached all of the scripts and prefabs we needed to get the player moving in our simulation.

2.2.4 Personal Computer (PC)

Throughout the development of this project we also had a PC version of the project. This version of the project was mainly used as a quick and easy way to add and test new features that we were planning to add to the Oculus build. It was also used to fix the many bugs that were created throughout the development of the project.

2.3 Using Version Control (Git)

Any project with more than one developer can be messy. To avoid this mess, we used Git.

2.3.1 Branches

During the development of this project we noticed many times that if you make a change within the Unity editor, it's common that over 20+ files have been changed within the project directory and this can lead to a lot of issues when it comes to pushing and pulling from the repository. This is because sometimes the same file has been changed twice before being pushed to the master branch, leading to a conflict. To alleviate this problem we used branches. This allowed us to give ourselves an environment in which we could create new features for the project without disturbing the master branch.

2.3.2 GitHub Issues

GitHub Issues was a good way to track all of the bugs in your project. It is also a very good way of reminding everyone working on a project what features/enhancements need to be implemented. We made use of labels to separate the issues into different categories (E.g. *unity*, *weather api/server*, *bug*, *enchancement*). This helped categorize and prioritise the issues. Through GitHub's issue auto-close feature, we closed some issues automatically through commit messages. This has the added benefit of linking a closed issue back to a commit, and vice-versa.

2.4 Testing

Throughout the course of the project, the way we went about testing was to manually test as we went along. Whenever we implemented a new feature into the project, we would test it to see if it fit the requirements. In hindsight, the use of an automated suite of tests could have been helpful in the development of our project, due to it's complex nature.

Chapter 3

Technology Review

The technologies that we used for the creation of this project each fall into one of two categories: *front-end* and *back-end*. The front-end technologies are what the user sees and interacts with, while the back-end technologies only exist to provide resources for the front-end.

3.1 Front-end Technologies

For the front-end, we decided to use Unity. Unity is a highly popular game engine which allows for the creation of games (or simulations, in our case) without having to develop a game engine from scratch. Doing so would have been a complicated and laborious process. Unity is also very effective at rendering both 2D and 3D scenes, has great cross-platform support, and is simple to use even without prior experience in game development. The cross-platform development aspect of Unity made it very easy for us to deploy our project onto the Oculus Quest, a virtual reality headset, as Unity SDKs are provided. All of the scripts we developed for the Unity project were written in C#. Unity projects are comprised of two main parts:

3.1.1 The Scene

In Unity, a *scene* is an arrangement of game objects. This should include at least one camera, which renders the scene for the user. In our simulation, the main scene consists of a house with a hallway, and two rooms attached to the sides. We designed this scene in a way that lets the user add new rooms dynamically. The user may walk around the house, opening doors or windows, and turning radiators on or off. As they do this, they can visualise the flow of heat in and out of the house, and between rooms. This visualisation is presented as temperature values changing on the minimap, and by the floors changing colour on a gradient, where red indicates hotter temperatures, and blue indicates colder temperatures. The functionality for these game objects is provided through the use of scripts.

3.1.2 The Scripts

In Unity, the *behaviour* exhibited by game objects comes from the scripts attached to them. In our project, the behaviour we required was mainly in the player's movement and interaction with the world, and the temperature exchange between rooms and around the house. Movement and interaction was made simple using Unity's built-in components for collision detection and physics, namely colliders and rigidbodys, respectively. In some areas, we were able to apply software development principles to write clean and efficient code. For example, inheritance was used between the radiator, window, and door scripts, to extract the functionality of a user being able to "trigger" these features when they click and are within a certain range (E.g. to open or close a window). This follows the "DRY" principle (Don't Repeat Yourself), and means that we avoided writing duplicate code.

Temperature Equalisation

For the temperature equalisation part of the scripts, it was important to represent the world in a data structure that made it easy to equalise the temperatures between the rooms and the outside, and between the rooms themselves. To achieve this, each room and each door has an adjacency list of rooms. This list is actually four individual references, pointing to the room that is north, south, east, or west. As such, the connections between rooms is essentially a 2D linked list. The reason that doors also have an adjacency list is to make it as easy as possible to equalise the temperature between two adjacent rooms. If only rooms had an adjacency list, it would be difficult to iterate over the rooms, since it is important not to equalise the temperatures of the same two rooms more than once (I.e. equalise the temperatures of room A and B, and then B and A). *Null* values are used where there is no adjacency.

For the purpose of our project, the equation for temperature equalisation is simple, although it does approximate the behaviour of heat in real life. One possible further exploration of this project is to make those exchanges as realistic as possible, making it useful as a teaching tool for thermodynamics students, for example. In all places where temperatures are equalised, the same general formula is applied: compute the difference in temperature between the two places, and take some percentage of heat from the hotter area and add it to the colder area. Doing this, larger temperature differences are settled quicker than smaller differences. The main areas where temperatures are equalised are:

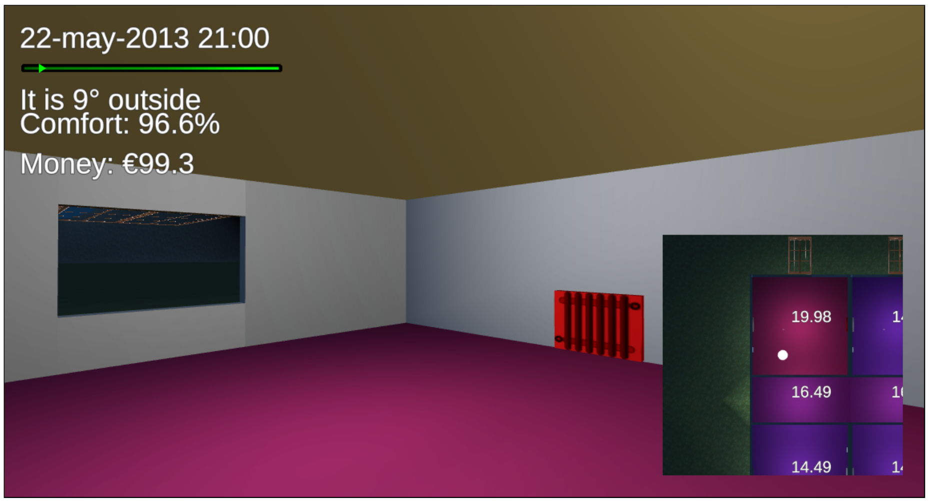
- Between a room and the outside weather
- Between two rooms
- Between two hallways

When a room's temperature is being equalised with the outside weather, the room's temperature simply approaches the current weather temperature using the method mentioned above. The outside temperature is not affected by the room's temperature though, since this difference would be minimal. The rate at which the room's temperature approaches the outside temperature is also affected by the number of walls which are exposed to the outside. This number is found easily from how many *nulls* are in the adjacency list. This temperature equalisation is also sped up if the window is left open by the player.

When the temperature of two adjacent rooms is being equalised, their two temperature values approach each other. If the door between the rooms is left open, this exchange happens much quicker. Our simulation also has a "special" type of room: a hallway. These rooms are joined by doors to adjacent rooms, but are joined by empty space to adjacent hallways. Because of this, hallways equalise in temperature with each other very quickly. The issue of iteration through hallways is very simple, since all of the hallways in our simulation run in a line and are attached to each other. This is why the trick of using doors to equalise temperatures between rooms was not needed for hallways.

To make the simulation more interesting, and to provide the player with a way of adding heat to the simulation, we also added radiators to each room. When the player interacts with them, they provide a constant source of heat to the room. To add a sense of realism, the radiators cut off at the twenty degree Celsius mark, turning yellow in colour to indicate this. Leaving radiators on also incurs a cost to the player, as shown on the UI. The UI also displays a player comfort value, influenced by the temperature of the room that the player is currently in. We included these features as they are a start into exploring the use of the simulation as an educational tool to highlight the cost associated with using different types of home insulation or radiator, for example.

Figure 3.1: A screenshot of the simulation showcasing various elements



3.2 Back-end Technologies

For this project, the role of the back-end is to provide historical weather data in an organized, "RESTful" way. The selection of weather data sets needed to be accessible in a list format. This would allow the list loaded into Unity in a way that allows the user to select a data set, and as such, the chosen data set could be retrieved from the server at another URL. It was also important to us that the data sets and protocol formats be easy to change without breaking the project, E.g. that adding more weather data sets did not require an update to the listings resource, since this resource should be generated per request.

3.2.1 Technology Stack

- API Architecture: REST
- API Description Format/Tools: OpenAPI and Swagger
- Framework and Database: Flask (Python) and MongoDB
- Formatting Weather Data: Python w/ Pandas
- Web Server Configuration: Nginx and uWSGI
- Server Hardware: Raspberry Pi

API Architecture: REST

Representational state transfer [6], or REST, is not a library or a tool. REST is an *architectural style* which describes a method of transferring data in a way that follows a set of rules. One of the most important rules is *statelessness*, which means that RESTful web services should not store any state with each transaction. This means that transactions are independent, allowing a lot of transactions to be cached. *Cacheability* is also an important REST property. It means that some resources can be temporarily stored so that the server does not have to keep generating the same response again and again. This reduces the load on the server, and can improve responsiveness. *GET* responses are a likely candidate for caching.

Another fundamental concept used in REST is the use of HTTP methods, or verbs. Instead of overusing one or two HTTP methods (usually *GET* for simple transactions and *POST* for transactions where a body is needed), RESTful web services make use of many different HTTP methods, each having their own unique purpose. These correspond to the four *CRUD* operations: Create, Read, Update, and Delete. The standard usages of these methods are as follows:

Operation	**Method**	**Action**
Create	POST	Creates a resource
Read	GET	Retrieves a resource
Update	PUT	Updates a resource
Delete	DELETE	Deletes a resource

These methods have different properties, as defined in RFC 2616.[7] A method is “safe” if it does not modify the resource, and therefore is cacheable. *GET* is an example of a safe method. Another property is “idempotency”. A method is idempotent if making the same request a number of times has no effect after the first request. For example, *PUT* is idempotent because updating a specific resource multiple times with the same data has no effect. *POST*, however, is not idempotent, because making the same request n times causes the creation of n new (identical) resources. These two properties have implications for how a REST API should be used. It should be noted that REST does not necessarily have to run over HTTP, however it is by far the most common protocol for use with REST.

REST was chosen as an appropriate method of data transfer for the weather data sets used in the project. Weather data sets are stored as resources on the web server, and can be accessed at specific routes. The routes are neatly organised in a way that allows a user to access different weather data from specific areas. Using the REST concept of *Hypermedia as the Engine of Application State*, or HATEOAS, weather data can also be listed, along with it's resource URL, to allow the user to browse data sets and select one that they like. This was done so that the user in the Unity simulation can select a data set on the UI in-game, so that they can experience different ranges of weather temperatures.

API Description Format/Tools: OpenAPI and Swagger

OpenAPI is a specification used for creating REST APIs. REST APIs can be defined using a markup language, E.g. YAML. This includes defining routes, data structures, response codes, and anything else needed to fully describe a REST service. While OpenAPI just specifies a format for creating REST APIs, Swagger can be used to define, implement, and maintain the service. Swagger is a set of tools built for use around the OpenAPI specification. The two tools we used to create our RESTful weather API were the Swagger Editor, and Swagger Codegen. The Editor was very useful for creating and maintaining the API specification. It has syntax highlighting, can detect errors in the specification, can display the API in a visual format, and much more. We only scratched the surface of what the Editor can do, as is the case with a lot of the technologies we used to create the project. This is because this software is geared towards large projects being worked on by many people, and in comparison, our project is quite small.

The other useful Swagger tool that we used was Swagger Codegen. Using a completed API specification, Codegen can generate client or server code "stubs" for a variety of languages and frameworks. For our project, we chose to generate a Flask server stub, which uses Python. Codegen can not infer the logic required to serve different REST routes. For example, it cannot know how to generate a list of weather data sets as a response when the client requests an overview of all of the weather data sets that are available from the Athenry weather station. The generated code for the routes is just skeleton code, but the models are fully defined. However, this still provided us with a quick and easy way to get started with the server code. We decided not to use Codegen to generate a client stub for the Unity client, since the client's role was so simple that it was easier to just make a web request to retrieve the JSON data for parsing.

Framework and Database: Flask (Python) and MongoDB

Flask is a web framework that uses Python. Using Flask, each REST route from our OpenAPI specification corresponds to a function. When a request is made for a specific data set, for example, code needs to be run which makes a further request to the MongoDB database for that data. The JSON data is then returned to the user, in the form of a HTTP response body.

MongoDB is a document-oriented database program. It is classed as *NoSQL* [8]. MongoDB is a popular program for storing JSON data, and is highly scalable. We chose to use MongoDB because it provided a quick and easy way of storing and retrieving the JSON weather data. We chose to use mLab for our web service, however in a more serious deployment, a local MongoDB database should be used to reduce latency. We considered just storing the JSON files on the local file system, which would reduce the complexity and possibly improve the performance of the service (since the JSON data could just be held in main memory on the server), but for the sake of introducing more technologies to our project and allowing the service to be scalable, we chose to use MongoDB.

Formatting Weather Data: Python w/ Pandas

Another important component to the web service is the weather data which it can supply. This data is stored in the MongoDB database, and is accessed by the service to serve each request. We chose to populate the database with real historical weather data from a weather station in Athenry. We started by downloading a large CSV file containing hourly weather data from the weather station on Met Éireann. Without modification, the file contained many fields which we were not interested in, such as atmospheric pressure. We were really only interested in temperature values for our project. The file was also extremely long, containing years of hourly weather data, and was not in the JSON format that our API required.

We created a Python script to split the weather data into a series of smaller data sets, while also converting the data into a JSON format and only including fields which were relevant to our project. Values can be adjusted to specify the weather simulation length in real minutes, the relative speed of the simulation compared to real time, the number of data sets to create, etc. Having these options helped us have full control over the data sets, and over their real time duration in the simulation. As per the API we created, some additional information is included in each data set file, including the length of the set in hours, and a brief description. We included an individual timestamp with each hour of data too, to make it easy for the simulation to display the current date and time on the UI.

Python has built-in data structures and libraries which make it very easy to create and manipulate JSON data. Specifically, Python's dictionaries, which work like maps in other languages, can be treated as JSON data and easily dumped to a file using the *json* library. In order to parse and manipulate the CSV data, we used a Pandas dataframe. Pandas is a powerful data analysis library, but here we make use of it just for the purpose of reading through the CSV file line by line, and reading the individual fields based on their column headers. Using the script we created, hourly weather data can effortlessly be converted to a format which corresponds to the API, and can be uploaded directly on the server for use.

Web Server Configuration: Nginx and uWSGI

Although Flask comes with a built in web server, it is not inadvisable to use this for production. This is shown as a warning when this built in server is used (it is for development purposes only). As such, a web server was set up to run the server application. Nginx (pronounced "engine-x") is highly popular, open-source web server software. It is efficient and lightweight, which was perfect for our simple REST API.

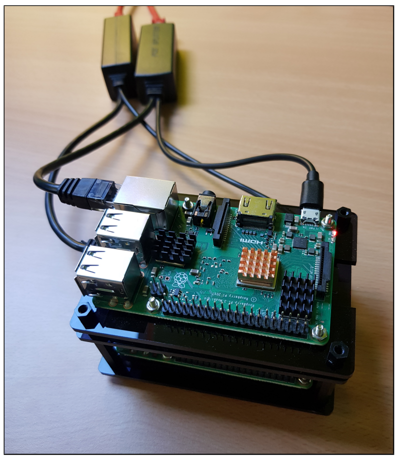
Along with Nginx, we installed uWSGI. uWSGI is a *Web Server Gateway Interface*. It works alongside Nginx, translating web requests to a format that can be dealt with by the Flask application. As a whole, Nginx, uWSGI, and Flask work together to provide the web service. Nginx accepts web requests, which it then passes to uWSGI, which then translates the request to a format that can be understood by Flask. Flask routes the request to the appropriate function, which then makes the appropriate request to the MongoDB database for weather data. A response is then generated, and is propagated all the way back to the client. We found setting up Flask, uWSGI and Nginx to be very tedious and difficult, but after some time of configuring those programs, the service is responsive and secure.

Server Hardware: Raspberry Pi

The last item in our technology stack is the device running all the above services: the Raspberry Pi. Developed as a way to give developing countries access to affordable computing hardware, the Raspberry Pi has also been a very popular choice for use in hardware and software projects. It is cheap, small, silent, energy efficient, and is more than capable of hosting a simple web server, such as the one we developed for this project. We used a Raspberry Pi 3 Model B+.

Raspberry Pi has an official operating system supported by the Raspberry Pi Foundation: Raspbian. Based on Debian, a massively popular Linux based operating system, Raspbian is tailored for use on Raspberry Pi devices. We did not need the desktop features of the OS (I.e. a desktop environment), since we just needed to host a basic web service. Because of this, the *"Lite"* version of Raspbian was installed. Using SSH and SFTP, we set up and configured the Pi for use with the aforementioned software. It is important for a device running a service available to anyone on the internet to be locked down and secure, so all the necessary steps were taken to ensure that the device was not vulnerable. This included setting up a restrictive firewall with UFW, ensuring a key file and password is used over SSH, and disabling root user access.

Figure 3.2: Raspberry Pi Model 3 B+ serving requests for our REST API



Chapter 4

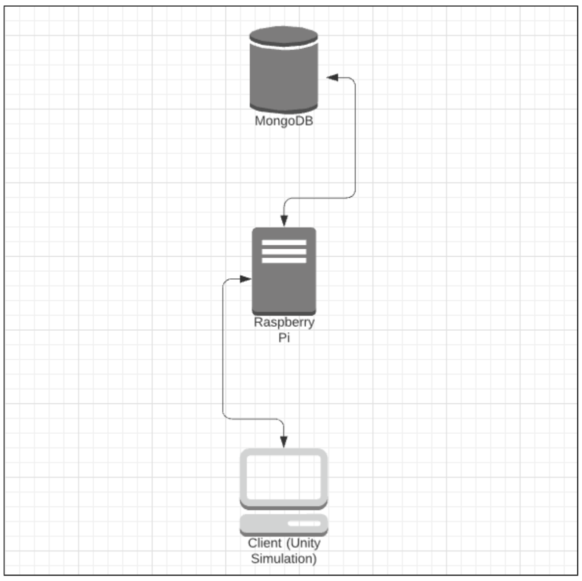
System Design

The system design covers how the technologies described in the technology review are incorporated into one overall system. The system design talks about how each component works, to link the overall system together.

4.1 Design Overview

The overall system design is composed of a client (Unity Simulation), which makes web requests to a server via an API, which talks to a server connected to a database.
This is how the client is able to talk to the database:

Figure 4.1: System Design Architecture



4.2 Server Architecture

4.2.1 API

Instead of creating scripts to format and parse the data into a format which would work well with Unity, we created an API to handle this. We processed and formatted data from Met Éireann into a format that suited us.
Here is an example of our API:

Snippet of weather data set listing:

```
[
  {
    "area": "athenry",
    "dataset": 0,
    "length": 240,
    "link": "https://<domain>:<port>/.../historical/athenry/0",
    "start_time": "15-aug-2012 09:00"
  }
]
```

... *(19 more data sets)*

Snippet of weather data set 0 (as shown above):

```
{
  "description": "Data set description here",
  "length": 240
  "data": [
    {
      "temperature": 17.1,
      "timestamp": "15-aug-2012 09:00",
      "windspeed": 16.0
    }
  ]
}
```

... *(239 more data points)*

4.2.2 Database (MongoDB)

We used MongoDB to store the various weather data sets for the project. When the Unity side is first opened, the user is greeted to a list of data sets. Depending on which one the user chooses, the client makes a web request to a different route of the API. Currently, we are using mLab to host the database, but future possibilities could lead to us having the database hosted on the Raspberry Pi itself. To get data from the database, the flask server makes a request for JSON data, which is returned to the client.

4.2.3 Flask Server

The Flask server acts as a mediator between the Unity Client and the database. The Flask server is hosted on a Raspberry Pi. It allows the Unity Client to make a web request to the server and get back the weather data in JSON format. We chose Flask over a framework like Java Spring (which we have also used during our course) because it is simpler and more lightweight, which is what we needed for such a simple API.

4.2.4 Web Server

The web server (Nginx) was used to translate web requests to a format that flask server was able to understand. Future applications of this web service could include generating random weather data for a random simulation and polling live weather data in real time.

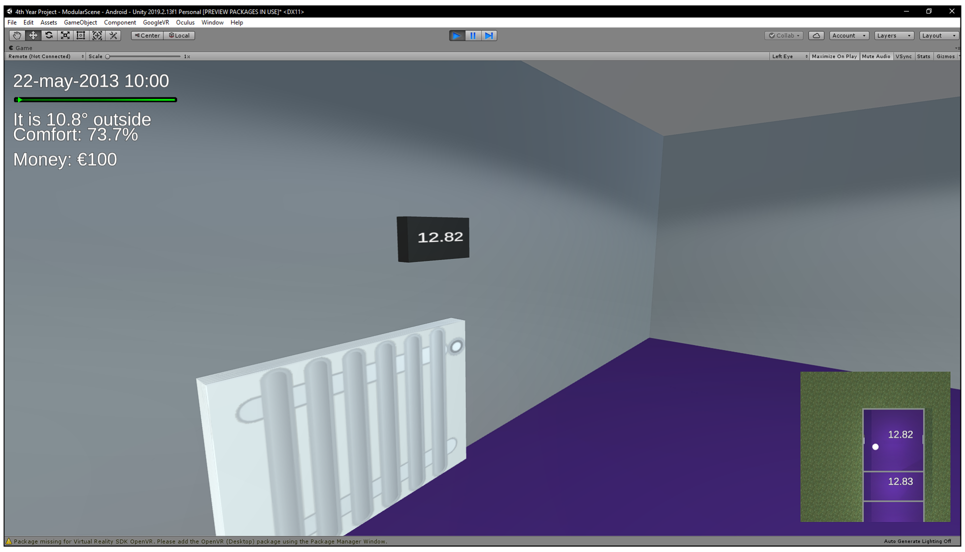
4.3 Raspberry Pi (Server Hardware)

The Raspberry Pi acts as the overall server which is running the Flask server and the web server all in conjunction.

4.4 Client (Unity)

The Unity side of the project was always going to be the main part of the project. The main aim of the Unity side of the project was to allow the user to navigate and manipulate a house in which each room's temperatures was being effected by the ever changing weather temperature outside, supplied by a web request to the web service.

Figure 4.2: User Interface (UI)



4.4.1 Breakdown of Unity Scripts

- **JSONReader.cs:** JSONReader.cs makes a web request to the REST API and gets back bytes from the server. It then deserialises those bytes into a String, which is then used to populate a WeatherHistory object. This WeatherHistory object is then used in the GameManager.cs to set the outside weather data for the simulation.

- **WeatherHistory.cs:** WeatherHistory.cs is an object that contains a list of DataPoints.

- **DataPoint.cs:** DataPoint.cs is an object that represents a single point in time. This single point in time has a timestamp value, temperature value and wind speed value.

- **Hallway.cs:** Hallway.cs has no core functionality. It serves as a reference point to a hallway GameObject.

- **Room.cs:** Room.cs is a representation of a room GameObject with respect to temperature. Room.cs is responsible for handling a rooms current temperature, the colour of the floor, the equalisation of temperature from one hallway to another and the warming up time of a radiator once it has been turned on.

- **AdjRooms.cs:** AdjRooms.cs (Adjacent Rooms) is a data structure representing up to 4 rooms that are connected. By connected, we mean they are beside each other.

- **RoomAdjUtils.cs:** RoomAdjUtils.cs (Adjacent Rooms) is responsible for setting up Room Connections. Room Connections are rooms that are beside the current room. This script is used by Instantiator.cs, to dynamically create connections between rooms at runtime.

- **DayNightCycle.cs:** DayNightCycle.cs is responsible for setting the Sun and Moon's starting position/rotation. It is also responsible for calculating the amount both GameObjects rotate per game step. DayNightCycle.cs is also responsible for setting the sky dome's offset to be the correct value for a particular game step.

- **Interactable.cs:** Initially, Door.cs, Radiator.cs and Window.cs all had relatively the same functionality so to prevent a lot of duplicate code. Interactable.cs was created to home all of this functionality in one place.

- **Window.cs:** Window.cs is responsible for rotating the Window Sprite the script is attached to when activated by the user. When a window is "activated" (opened), the temperature equalisation to the outside world is sped up.

- **Door.cs:** Door.cs is responsible for rotating the Door Sprite the script is attached to when activated by the user. Door.cs is also responsible for equalising the temperature between the rooms it is connecting. Door.cs is also responsible for increasing/decreasing the rate at which temperature equalisation occurs depending on if the user has activated it or not.

- **Radiator.cs:** Radiator.cs allows radiators to be switched on or off by the player. It also contains the code that heats the room up over time. A realistic kilowatt-hour calculation is also performed here to calculate the cost that should be incurred to the player.

- **GameManager.cs:** GameManager.cs is responsible for the overall management of the simulation. Every update, using the downloaded weather data, it orchestrates the temperature exchange for every room within the house. GameManager.cs also renders most of the UI elements for the simulation. A lot of the initialisation code for the house is also done here.

- **Instantiator.cs:** Instantiator.cs is the one of the most intricate scripts of the whole Unity project. It allows the player to dynamically add rooms and hallways to the house. Using RoomAdjUtils.cs, the connections between the rooms for temperature equalisation are also linked up.

- **WalkWhereLooking.cs:** WalkWhereLooking.cs was created for use with the Google Cardboard VR deployment of the project. It allows the player to move around the house by looking in the direction which they want to go in, and touching the screen (done using the Cardboard's included "button").

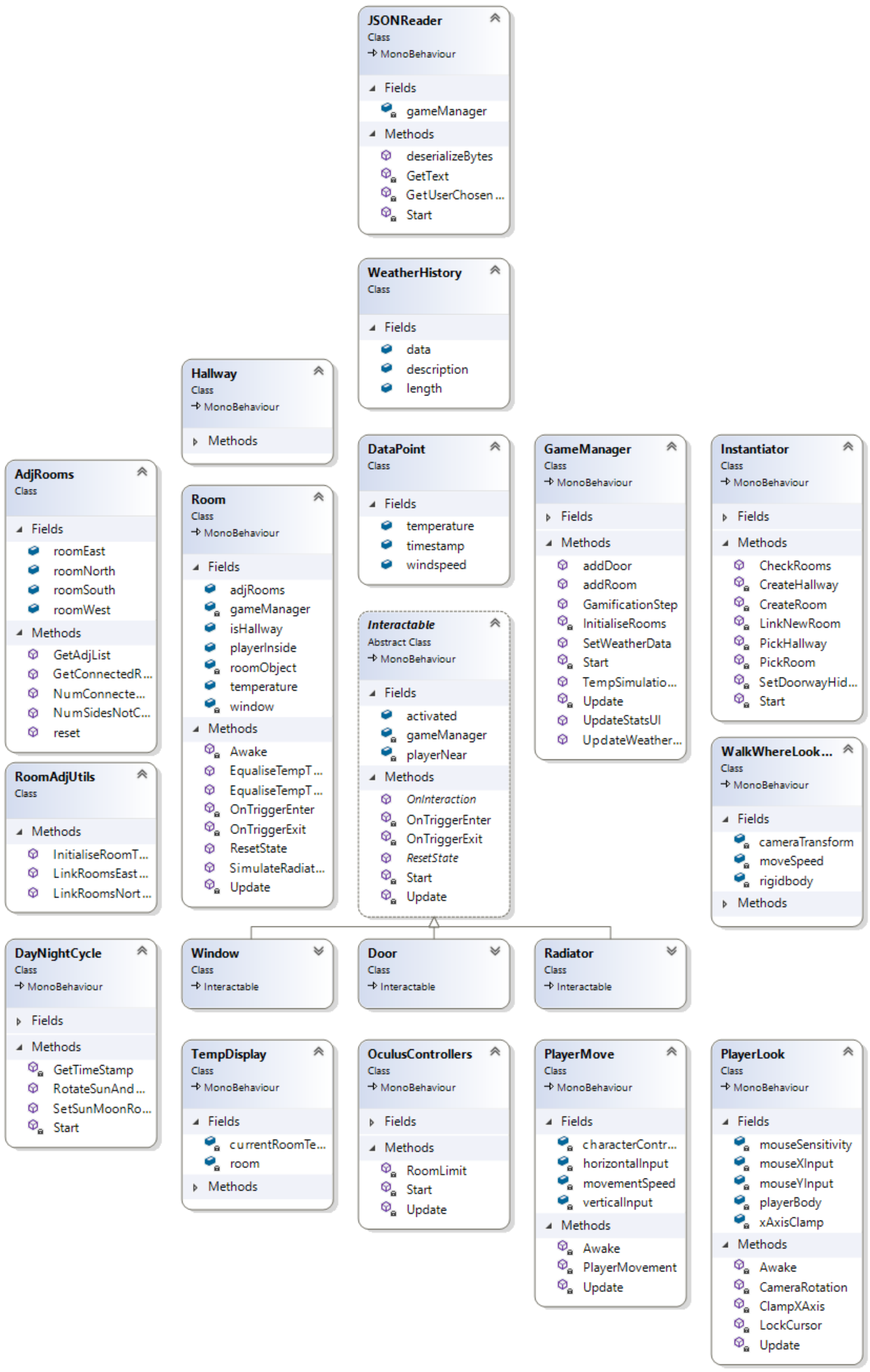
- **TempDisplay.cs:** TempDisplay.cs is responsible for drawing the temperature value of each room onto the minimap. A similar script, TempDisplayRoom.cs, draws each room's temperature on the thermostat placed in each room.

- **OculusControllers.cs:** OculusControllers.cs contains extra functionality for use with the Oculus controllers. One of these functions is allowing the player to spawn rooms when using the Oculus deployment of the simulation.

- **PlayerMove.cs:** PlayerMove.cs is used on desktop deployments to allow the player to move around the house, using the standard W, A, S, and D keys. The player moves at a fixed speed in a direction corresponding to which of those keys is pressed.

- **PlayerLook.cs:** PlayerLook.cs is attached to the main camera on desktop builds of the simulation, and allows the player to move the camera using the mouse. The rotation angle is clamped to restrict the player to a natural field of view.

Figure 4.3: UML Diagram



Chapter 5

System Evaluation

5.1 Performance

5.1.1 Temperature Logic Performance

As video games are typically expected to render at least 60 frames per second to keep up with the standard 60 hertz of a computer monitor, games usually also have a high number of logical updates per second. During these updates, input has to be handled, collisions have to be detected, etc. The amount of time Unity has to perform an update (in seconds) is at most:

$$\frac{1}{updates\ per\ second}$$

If Unity is set to update 30 times per second in the Editor, for example, the update should execute in under **33.33 milliseconds**. Any longer, and the game would begin to slow down or even freeze up. This shows that, by their nature, video games **must** be highly efficient if they are to run smoothly, even on modern hardware. Simplifications, approximations, and various tricks are commonplace in game development to prevent the game from slowing down to a halt.

The temperature logic for our game is the most critical area in terms of performance. The user should be able to create many rooms, open windows or doors, and turn radiators on or off etc, without causing the game to slow down. In order to allow this, we created a data structure connecting rooms and doors. The specifics of this data structure have already been discussed in the Temperature Equalisation section of the Technology Review chapter.

The main area of the code where the temperature logic is updated is in the GameManger's **TempSimulationStep()** method. The number of minutes that have passed since the last temperature update was performed can be passed in as an argument. We did this to allow the heat to flow around the house in *real time*; originally the temperature just updated at fixed intervals, I.e. every second. Real time temperature equalisation is much more visually impressive, but as a trade-off, is more resource intensive because of the high number of updates required every second. Implementing real time temperature equalisation required efficient code.

Performance Breakdown

TempSimulationStep() contains four loops. Each loop implements the temperature logic for one particular feature of the house. Here is the overall algorithm, with notes about time complexity:

1. For each room in the house, equalise heat with the outside world.
    - Since the operation performed for each room runs in **constant time**, essentially just evaluating a mathematical expression, performing this for all rooms runs in linear time, or $\mathbf{O(n)}$, where n denotes the *number of rooms in the house.*
2. For each room, add the appropriate amount of heat for each activated radiator in the room.
    - Looping over each radiator in each room is an operation which runs in $\mathbf{O(nm)}$ time, where n denotes the *number of rooms*, and m denotes the *the average number of radiators in each room.* Since we modeled all side rooms in our game to have two radiators, a constant number, this is effectively reduced to $\mathbf{O(n)}$.
3. For each hallway, equalise heat to the hallway to the east, if there is one.
    - Similarly, since equalising heat between two areas is a constant time operation, this runs in $\mathbf{O(n)}$ time, where n denotes the *number of rooms in the house.* Any rooms in the list which aren't hallways are ignored. To optimise this further, a separate list of only hallway rooms could be maintained, although in practice the improvement in performance would be minimal.
4. For each door, equalise heat between the two connecting rooms.
    - As with the other operations, equalising heat between the two rooms adjoined with each door is done in constant time. Therefore, doing it for each door is done in $\mathbf{O(n)}$ time once again, where n denotes the number of doors in the house.

As shown, temperature equalisation operations across house as a whole is a *linear* time operation, since **each of the four operations effectively runs in linear time**, growing with the number of rooms, hallways, and doors. Here, big O notation can be seen to be a useful metric in measuring the performance of the code. For example, if the temperature equalisation logic worked by equalising the heat between *every room* and *every other room*, the time complexity would rise to an order of $\mathbf{O}(n^2)$, where n is the number of rooms. The performance of the game would then slow down rapidly with each room the player spawns in, which is clearly not desirable.

5.2 Limitations

Temperature Logic and Room Configuration

Because of the way in which we constructed the house and the underlying temperature logic, there are of course some limitations as to how things can be arranged. Any number of radiators can be placed in any of the rooms, including the hallways. To simplify the temperature exchange process between each room and the outside world, each room can only have up to one window. Allowing multiple windows per room would be possible, but this would require some changes to the Room script. The formula that decides how fast the heat in the room would equalise to the outside would have to take into account how many windows are open in the room. Also, changes would have to be made to how new rooms with more than one window are spawned in, making sure that new the room is orientated correctly so that its windows don't collide with already existing rooms.

As for the arrangement of doors and rooms, our data structure allows for each room to be connected to up to four other rooms, through the placement of doors, which must be connected to exactly two rooms. Although the code and Unity objects refer to rooms as being north/south/east/west from each other, in practice these don't have to be strictly followed. This can be seen in our use of the L hallways, which connect to two hallways and two side rooms in a staggered configuration. There is room in our implementation for the creation of other configurations of rooms and hallways, such as this. Also, allowing any number of rooms to be connected to each room would be possible, if the cardinal directions we used to define connections between rooms were removed, and the underlying adjacency list was used instead. This would allow the creation of a very long hotel corridor, for example. We got around this problem by breaking down long hallways into smaller rooms in our implementation.

REST API

One of the drawbacks we experienced with using a REST API over HTTP was that we believed it was not a suitable platform for streaming data in a complicated setup involving multiple devices. One of the original ideas for the project suggested to us by DXC in their initial project specification email was to make use of an *Internet of Things* (IoT) device to stream live temperature data from the real world into the simulation. This could give an extra layer of realism to the simulation, and allow the user to experience and manipulate real world temperature data.

Soon after setting out to implement this functionality, however, we realised that using the REST API for this was probably not appropriate. We wrote up a plan for how this could work over gRPC instead. gRPC is a high performance remote procedure call framework which we had learned about and used in another module. This plan, including a mock-up protocol, can be viewed on the git repository associated with this project under *research/sensor-data-streaming.md*. Along with complications with the IoT device we were trying out from the college, though, we decided to forgo implementing this idea for our project.

5.3 Completion of Objectives

Soon after we began working on the project, we came up with a list of our overall goals which we wanted to achieve. These original goals can be viewed on the **Project Scope & Plan** page on the wiki for our project on GitHub. Some goals were completed fully, some were partially completed, and some were ignored due to time constraints. These were the goals we set out to achieve:

- **Implement the player (movement, interactions, etc.)**

  We of course **completed** this goal, which was central to the simulation. Inside the simulation, the player...

  - Views the world using a first-person perspective camera
  - Can move around the house using the keyboard or Oculus controls
  - Collides with solid objects, to restrict the player's movement to within the house/room they are situated in
  - Can interact with features of the house, including windows, doors, and radiators

- **Make game objects (rooms, outside, radiators, windows, doors, thermostat etc.)**

  We created and made use of all of the features listed in this goal. We also created two different types of hallways, lighting features, terrain and skybox features, and a rotating sun and moon.

- **Create the server (local/cloud)**

  As discussed in the **Back-end Technologies** section of our **Technology Review**, we successfully made use of a web server to provide historical weather data in a machine friendly format. This server does not run "locally", or in the "cloud", since it runs on a dedicated device from home, but this isn't really a critical aspect of the goal.

- **Create the database (MySQL)**

  We made use of MongoDB, which is document-oriented database software, on our web server. Although we initially planned on using MySQL, we soon realised that MongoDB is more suited to storing and serving JSON documents. Again, we achieved this goal, but in a slightly different way from which we had planned initially.

- **Create the API (static)**

  Making use of the OpenAPI standard and Swagger tools, we created a simple, robust REST API.

- **Populate the game objects with data from a file / Populate the file in the Unity project**

  Although we had plans to load the weather data from the REST server into a file and then into the simulation, we moved away from the idea of using files and instead just manipulate the weather data from within memory. As such, these goals are no longer relevant. Despite the wording of these goals, we still achieved the functionality which they are getting at: making use of the weather data inside the simulation.

- **Implement the temperature logic (basic)**

  Making use of the right data structures, algorithms, and formulas, we successfully created an approximated model of temperature flow through a house. The player can manipulate the various features of the house (doors, radiators, and windows) to influence the flow, dissipation, and generation of heat.

5.3.1 Stretch Goals

Here are some of the additional goals which we were hoping to complete for the project:

- **Adding heaters**

  We added radiators to each room, completing this goal. Radiators can be toggled on or off, providing the player with a way to add heat to the simulation. Prior to adding radiators, the house would just gradually approach the outside temperature.

- **Turn it into a training simulation, or "gamify" it**

  We introduced money and player comfort in an attempt to add "gamification" elements to the simulation. The player can visualise how much money they spend over time if they leave some radiators on in the house. This is based on realistic kWh calculations. The comfort value also gives some indication as to whether a real person would feel comfortable being in a room at a certain temperature. We talked about expanding on this further: maybe the objective of the game could be to keep the player at a comfortable temperature for a certain amount of time, without running out of money, etc. This goal was **partially completed**, leaving this idea ready to be explored more with further development to the project.

- **Temperature logic (advanced/realistic)**

  Although this is a feature that we thought would make the simulation a lot more interesting, we unfortunately ran out of time to implement it. With this feature, the aim of the project could be to experiment with the real world properties of different materials as insulation, for example. It could also be used to show the economical benefits of using a modern radiator instead of an older one. These ideas show that implementing realistic temperature logic for the simulation could open it up for use in a variety of interesting applications.

- **Using a sensor to gather real word data, streaming that data into the simulation**

  Unfortunately, due to some complications with the IoT hardware provided by the college, we decided we did not want to pursue this goal any further.

5.3.2 A Reflection on our Goals

As can be seen above, we completed all of the fundamental goals that we set out to complete for our project, and some of the stretch goals too. Since these goals were based on the project specification and suggestions given to us by DXC, we feel that we have successfully completed the project which was proposed to us, and to the standard expected from 4$^{\text{th}}$ year students at NFQ level 8. We have also left the project in a state where it can easily be picked up for further development and exploration of the project themes mentioned.

Chapter 6

Conclusion

Through extensive use of a HMD over the past few months, we can safely say that there is a lot of potential when it comes to using a HMD as a medium for learning/training.

6.0.1 Evaluation

In our evaluation section, we showed how we successfully made use of efficient data structures and algorithms in order to represent live temperature equalisation within a house. Despite the performance implications of running these operations many times a second, our implementation held up, and the simulation runs smoothly on system with reasonably modern hardware.

As with any software project, our project has limitations in every aspect. Many of these limitations lie in the configuration of rooms, hallways, doors, and radiators. Looking back on the project after we finished developing it, we are happy with the sacrifices we made in that regard. The user has sufficient choice in how the house is configured. We also discussed how our REST API has limitations, notably that it can not readily stream weather data. Further development in that area would open up the possibility of streaming live data into the simulation, to visualise it's effects.

6.0.2 Future Opportunities

After our development of the project, it is clear to us that the project has potential in a number of areas. We discussed some of these ideas in the beginnings of the project. In some areas of the project, we purposely began exploring these areas, to allow the project to be expanded on in the future. Here are some of the ideas we had for future expansion of the project:

Gamification

As mentioned in the introduction, the gaming industry is massive. By adding win/lose conditions, and some sort of score value, the simulation could be turned into a game-like simulation. Changing the simulation into a game could improve user engagement through the use of a reward system. For example, a ranking table sorting players by who had the highest comfort percentage for the longest time.

Demonstration of Home Heating Costs

Many people today don't actually know how much it financially costs to heat a home. A future adaptation of this project could include variables that help in the demonstration of the cost of heating a home. For example the rooms could include insulation in the walls to decrease the rate at which a room cools, which ultimately reduces the cost of heating that room. This could also demonstrate the cost of heating a home at different times in the year. Heating a home to a certain temperature in the Summer tends to be cheaper than in the Winter, due to increased air temperatures. Future adaptations of the project could also include solar panels. Solar panels would be a great way to demonstrate that investing a large amount of money in a short space of time could save an individual a large amount of money that would have been spent on heating in the long term.

Thermodynamics Education

For people wanting to learn about thermodynamics, this simulation could be used as a learning aid. To achieve this, the temperature equalisation code would have to be changed to a more realistic model, as right now it is just an approximation. Instead of the simple model we used, the properties and thickness of each of the materials used in-game would have to be taken into account, alongside the use of complicated mathematical expressions and equations. If this was done, however, the simulation would surely be of great use to students as a teaching tool.

6.0.3 Final Thoughts

When we started this project, the main concept was to demonstrate the effectiveness of using a virtual reality headset as a medium for learning. Throughout months of development, we successfully placed the user into a virtual environment that represents the flow of temperature within a house.

Overall, we both enjoyed developing the project under the guidance of DXC and our project supervisor. It was a great opportunity for us to collaborate on the creation of a software product that was completely different to anything else that we were required to produce in the other modules we participated in during the course. It was very interesting to see Unity being used for the purposes of an educational simulation instead of a simple game.

We are both excited to see what future adaptations of a project like this can bring to the education and training sector. We are hopeful that, through this project, we were able to showcase the great potential of Virtual Reality as tool for education and training.

Bibliography

[1] N. Bates-Brkljac, *Virtual Reality.* Computer Science, Technology and Applications, Nova Science Publishers, Inc, 2012.

[2] M. J. Wolf, *The video game explosion: a history from PONG to Playstation and beyond.* ABC-CLIO, 2008.

[3] J. Funke, "Computer-based testing and training with scenarios from complex problem-solving research: Advantages and disadvantages," *International Journal of Selection and Assessment*, vol. 6, no. 2, pp. 90–96, 1998.

[4] M. Prensky, "Digital game-based learning," *Computers in Entertainment (CIE)*, vol. 1, no. 1, pp. 21–21, 2003.

[5] S. Coştu, S. Aydın, and M. Filiz, "Students' conceptions about browser-game-based learning in mathematics education: Ttnetvitamin case," *Procedia-Social and Behavioral Sciences*, vol. 1, no. 1, pp. 1848–1852, 2009.

[6] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures (Chapter 5).* PhD thesis, University of California, Irvine, 2000.

[7] I. N. W. Group, "Hypertext transfer protocol – http/1.1."

[8] B. Jose and S. Abraham, "Exploring the merits of nosql: A study based on mongodb," 2017.