EDUCATIONAL SIMULATOR WITH REALTIME DATABASE

BY

TOK ZHI SUNG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER ENGINEERING (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

JANUARY 2019

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: ____

____

____

Academic Session: _____________

I ____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

	Verified by,
_________________________	_________________________
(Author's signature)	(Supervisor's signature)
Address:	
__________________________	
__________________________	_________________________
__________________________	Supervisor's name
Date: _____________________	**Date**: ____________________

EDUCATIONAL SIMULATOR

WITH

REALTIME DATABASE

BY

TOK ZHI SUNG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER ENGINEERING (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

JANUARY 2019

DECLARATION OF ORIGINALITY

I declare that this report entitled "**EDUCATIONAL SIMULATOR WITH REALTIME DATABASE**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	____________________________
Name	:	____________________________
Date	:	____________________________

ACKNOWLEDGEMENTS

I would like to express my very great appreciation to Dr. Chang Jing Jing for her valuable and constructive suggestions during the planning and development of this project. This is a completely new field in my research. I learned a lot under her guidance. Her willingness to give her time so generously has been very much appreciated.

Finally, I wish to thank my parents and my family for their love, support and continuous encouragement throughout my study.

ABSTRACT

A heat exchanger is a device used to transfer heat from one medium to another. It is widely used in different fields. Air conditioning is the most common example of everyday life. Experiments on heat exchangers have been conducted by engineering students. However, the time or cost of using this equipment is high. Therefore, people are looking for multimedia software to replace this teaching equipment, especially SOLTEQ, which is the manufacturer of engineering education teaching equipment. In response to the company's needs, the purpose of this project is to create a simulator for the heat exchanger.

The Unity game engine and Google Firebase are the main tools for creating this simulator. The Unity game engine is a tool for creating own 3D objects. The properties of objects created in Unity are customizable. Firebase can be imported into the Unity project to bring more functionality. Therefore, the merger of Unity and Firebase was done in this project and produced the final product - a heat exchanger simulator with realtime database.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SDK	Software Development Kit
IT	Information Technology
2D	Two-Dimensional space
3D	Three-Dimensional space
NoSQL	Non-Structured Query Language
API	Application Programming Interface
UID	Unique user Identifier
UI	User Interface
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
UML	Unified Modelling Language
URL	Uniform Resource Locator
JSON	JavaScript Object Notation

CHAPTER 1 INTRODUCTION

1.1 Problem Statement and Motivation

1.1.1 Problem with Existing Heat Exchanger Unit

In engineering education, the heat exchanger unit is a useful teaching equipment for engineering students to learn and investigate the fundamental principles of heat transfer. However, this equipment is not only huge in size but also extremely expensive in cost. Many colleges or universities found it is not affordable to purchase such equipment just for educational purposes only. They also found that such equipment is not very efficient to use because it typically takes a lot of time to set up before one can use it. In addition, there are only a few students can use it to carry out the experiment in one time. As a result, people are looking forward to the solution such as an alternative way to substitute this equipment.

1.1.2 The Use of Database to Record Experimental Data

During the experiment, students usually need to collect the data for further analysis. However, it is inefficient to record down all the data on the paper, especially for the dynamic system since the data is varied from time to time. Data analysis will become difficult and complicated when there is too much data. Rather than recording the data on paper, it is better to store all the data into a computer. With this idea, the database is used as a tool to store the data, which can provide an effective way for analysis and bring advantages. However, creating a database is cumbersome and difficult work. It needs a lot of effort to develop it. Fortunately, with the presence of Firebase (cloud database found in September 2011), it reduces the database development workload. However, there are limited resources to practice or refer to since it is a new technology in the market that merges with Unity. The first Unity Firebase SDK was released on November 7, 2016.

1.1.3 The Motivation for Creating a Simulator

In the industrial world, computer simulations are often used to conduct dangerous or impractical experiments in the real world because it is not only an alternative way to study the real-world problems in a safe environment but also a convenient way to reduce the cost and time in practice, thereby improving the efficiency of experiments. Thus, a heat exchanger simulator will be created in this project. It aims to help engineering students or industrial operators to equip up the knowledge of heat transfer without the need to operate with real equipment.

1.2 Project Scope

At the end of this project, an educational simulator with real-time database for the heat exchanger system will be created. The followings are the scopes of this project:

- SOLTEQ shell-and-tube heat exchanger will be used as the model
  In this project, the model of heat exchanger used is model HE104-4, which is a shell-and-tube heat exchanger unit produced by the SOLTEQ Company. The company provides specifications and details of the heat exchanger, which will be used as a reference for developing the simulator.

- The mathematical model is adapted from the existing model
  When modelling the heat exchanger system, the scope involved can be expanded indefinitely to cover as many situations as possible in order to approximate the real model in the physical environment. However, to consider all the possible situation is impractical and it is not in the field of information technology (IT). Hence, this project will directly use the modelling technique proposed by other researchers from heat transfer engineering (Daniel J. Correa & Jacinto L. Marchetti), eliminating the need to create own mathematical model for the heat exchanger system.

- The Firebase real-time database is used
  Firebase offers two cloud-based databases: Real-time Database and Cloud Firestore, both of them are client-accessible database solutions that support real-

time data syncing. Currently, the latter is still in a beta version. In order to avoid mistakes, it is better to choose Firebase's original database (the former – Real-time Database).

1.3 Project Objectives

The objective of this project is mainly concerned with 2 points:

- To develop a mathematical model for a heat exchanger simulator
  It is difficult to form a mathematical model for the heat exchanger system because of its high degree of complexity. One needs much effort and skill to accomplish, deal with, or understand it. Usually, the mathematical model of a heat exchanger system formed by most of the researchers is complex partial differential equations. It is not easy to encode such equations into a programming language that does not have multi-paradigm numerical computing capabilities. Therefore, this project will figure out the simplified equations so as to facilitate the calculations using the C# programming language and finally introduce them into the Unity simulation.

- To incorporate a real-time database to the simulator for ease of data accessing
  As mentioned earlier, the students usually need to record all the data during the experiment. A real-time database would boost the recording of experiment data, thus making the accessing of data (such as store, retrieve, move or manipulate stored data) easier and facilitating the further analysis.

1.4 Impact, Significance and Contribution

In order to control the behaviour of "GameObjects" in Unity (where the GameObject is a heat exchanger), the script must be created and attached to the object. However, it is worth noting that the behaviour of heat exchangers is often described as a complex partial differential equation, which is hard to calculate in C#. Therefore, the first contribution of the project is to convert the mathematical model in partial differential equations into a more suitable form for scripting that can be implemented in C#.

Apart from that, the next contribution is to explore the method of using Firebase in Unity. Dating back to 7th November 2016, the first version of Firebase Unity SDK was released. Since it is a new form of cloud database for Unity game developer, this project will explore the potential of firebase and use it to enhance the simulator.

1.5 Background Information

First, what is a heat exchanger? Just like its name tells, it is a process equipment dedicated for transferring heat from one element to another in order to heat up the system or cool it down. They have been widely used in various fields such as homes, workplaces, and especially industrial fields. In most of cases, cooling is its primary function so as to avoid equipment from overheating which may damage or even destroy the equipment. In many industrial processes, it is required to keep a certain degree of temperature to ensure the functionality of a system. Otherwise, the stability of the system will be affected and turn down the performance of the system directly or indirectly. For instance, a heat exchanger is utilized to keep synthetic compounds, gas, hardware and different substances inside a safe working temperature so it won't result in desperate outcomes. Therefore, it is very important to study and develop a well-performed and satisfactory heat exchanger in the industrial field. Simulators are always used as the tools for studying heat exchanger.

Second, what is a computer simulation? Computer simulation is different from the genuine experimental method, which is generally used to investigate the working state of the system and has incredible restrictions. It can do the conceivable, conservative and advantageous constrained test before the establishment of the genuine system. Computer simulation is used to carry out experiments which are impossible or impractical. It has become another way to reduce costs and save time as compared to traditional laboratory experiments. It helps people to solve the real-world problems in a safe and efficient environment.

There is a lot of simulation software available in the market, such as MATLAB, ANSYS, Dymola and so on. This kind of software often require people to purchase licenses from them, otherwise people can only use limited features with other more useful features are not accessible. Moreover, their sophisticated interface is not easy for a beginner to use. Instead, Unity is a cross-platform game engine designed to create 2D and 3D games and simulations. It can be used to create customized GameObjects and user interfaces, so it becomes a tool for visualizing real objects in the world. In other words, it can be used to create a customized simulator.

As a consequence, a heat exchanger simulator will be proposed and created in this project. Students can utilize it to conduct an experiment and learn the knowledge of heat transfer. In addition, the Google Firebase – a NoSQL database will also be included in this project for further exploration and adding to the simulator in order to facilitate the work.

CHAPTER 2 LITERATURE REVIEW

2.1 Heat Exchanger System

The heat exchanger is a piece of equipment to transfers thermal energy from a liquid to go to a second liquid at different temperatures without mixing them up (Woodford, 2018). The basic working principle of a heat exchanger can be simplified as shown in Figure 2.1.1

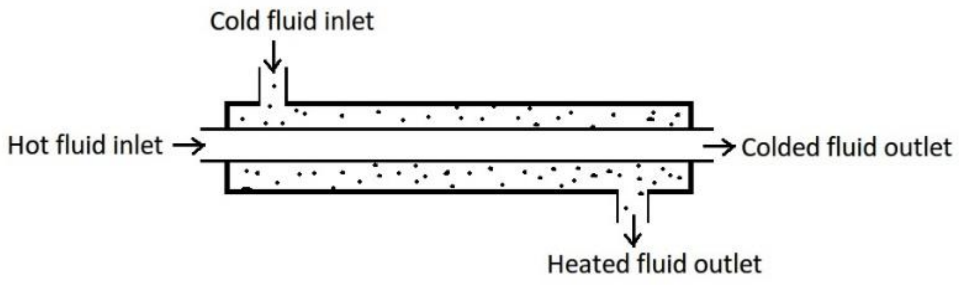


Figure 2.1.1: Basic working principle of a heat exchanger.

Generally, the heat exchanger consists of many thin tubes running through a large cylindrical shell. Figure 2.1.1 is just a simplified example of a shell-and-tube heat exchanger. When a hot liquid flows through the tube inside, heat is transfer to the cold liquid (shown in dotted) at the outer tube. Hence, the hot liquid cools down and the cold liquid warms up without the liquid direct contact and mixing them up.

In order to fulfil a variety of different situations and requirements, heat exchangers can come in different forms. The heat exchangers are classified into different type according to transfer processes, construction features, flow arrangements, the degree of surface compactness, the number of fluids and heat transfer mechanisms (Shah & Sekulić, 2007).

Among the various kind of heat exchangers, shell-and-tube heat exchangers and plate heat exchangers are the most widely recognized heat exchangers in the modern field, where the shell-and-tube heat exchanger unit is provided for reference and study in this project. As compared to the plate type heat exchanger, the shell-and-tube heat exchanger has more advantages. Below is the summary of its advantages and disadvantages:

Advantages:

- Less expensive.
- Withstand higher working temperatures and pressures condition.
- Due to the relatively easy pressure testing, leaking tubes are easily located.
- Using a sacrificial anode can protect the entire cooling system from corrosion.

Disadvantages:

- Possible results in clogging due to the pathways are very small.
- Hard in terms of the cleaning process.

In addition, a few researchers in the engineering field also agree that shell-and-tube type exchangers are popular in the process industry and can be easily modified in most cases (LUNSFORD, 2016). For this reason, the research in this area appears to have importance.

2.2 <u>An Overview of Heat Exchanger Modelling</u>

Since the start of the computer era, an ever-increasing number of physical phenomena have been modelled with a specific end goal to simulate as opposed to conducting trials. Trials are regularly exorbitant because they require the need for exploratory setups, like physical components and measuring apparatus. On the other hand, simulation only requires a computer, a simulation tool, and a model. The computers are available everywhere nowadays. However, the merging of the simulation tool and model for simulating heat exchanger systems are technically challenged. This is because the people who are good at developing simulation tools do not necessarily understand the concept of the subject model, and those who develop models do not necessarily know how to develop simulation tools.

There were many people had simulated and modelled the heat exchanger with different approaches. For instance, the previous work used Computational Fluid Dynamics packages ANSYS 13.0 (an engineering simulation software) to solve the modelling and meshing of the basic geometry of shell-and-tube heat exchanger (Sunil, et al., 2014). In spite of the fact that ANSYS provides a variety of powerful features for creating a simulation, the interface is very complex and the result is that it is not easy for novices to use. In addition, ANSYS is also known as a commercialized software, developers cannot arbitrarily modify the software due to copyright. It also included another package that no need in the simulation and causing redundancy.

Apart from that, John Hellborg modelled heat exchanger using Dymola (Hellborg, 2017). Dymola is a commercial tool for modelling and simulation based on the Modelica modelling language. The Dymola had also been used to visualize the simulation of control education (Martin-Villalba, et al., 2009). Their motivation was to encourage the client's intuitive activities on the model. However, Dymola is not an ideal tool for visualization purpose. It only provides two basic ways for the visualization of simulation results: plotting and animation (Martikka, 2004). Furthermore, the researchers said that “the visualization term of this work is poor and it seems that it is not enough attractive for student”, further stated that Dymola is not useful for visualization (Amirkhani & Nahvi, 2016).

2.3 <u>The Development of Mathematical Model for Heat Exchanger System</u>

There is a lot of ways to form the mathematical model. A straightforward way is to use a static model to compute the final variable, which is the final outlet temperature of the heat exchanger. Another way is to use a dynamic model by incorporated differential equations. The first method calculates the system in equilibrium and it fails to capture their state evolved with respect to time. The second method is more complex to compute as it considers for the time-dependent changes, but it is more informative (Fritzson, 2004). In the simulation, it is preferred to use a dynamic model so as to observe transient behaviours.

Many researchers have developed dynamic models for the heat exchanger to analyse its transient behaviour. Different methods have been studied. The transient response of a counter-current double-pipe heat exchanger was discussed in (M.R.Ansari & V.Mortazavi, 2006). The study has been carried out to test the response of counter-current double-pipe heat exchanger. The method used was to merge the numerical method with analytical methods. The results took into account the accuracy of the mathematical calculations and the amount of computation time. In addition, the comparison between finite-volume and moving-boundary formulations for shell-and-tube heat exchanger were also studied in (Bendapudi, et al., 2004). The finite-volume approach offers more detail yet at critical computational cost, while the moving-boundary approach takes less time. These studies provide some good methods for dynamic modelling of heat exchangers, but they do not provide a calculation method for digital computers.

A modelling technique which could be applied to the dynamic simulation of digital computer, where it could be used into almost all kind of shell-and-tube heat exchanger, had been proposed by C&M (CORREA & MARCHETTI, 1987). The mathematical model is first described as partial differential equations but was later simplified into ordinary differential equations, and was further simplified into algebraic equations. They also proposed the approaches for solving the algebraic equations by applying an iterative procedure. For this reason, their proposed method can be further implemented in programming language feasible.

The model is extended from the concept of previous researchers (GADDIS & SCHLüNDER, 1979). Refer to the figure 2.3.1, they suggested modelling the multi-pass shell-and-tube heat exchanger by dividing it up into several cells. The quantity of these cells i = 1, 2, ⋯; NM indicate the arrangement of cells following the tube-side liquid direction beginning from the entrance. N is the quantity of baffles in the shell, and M is the number of tube passes.

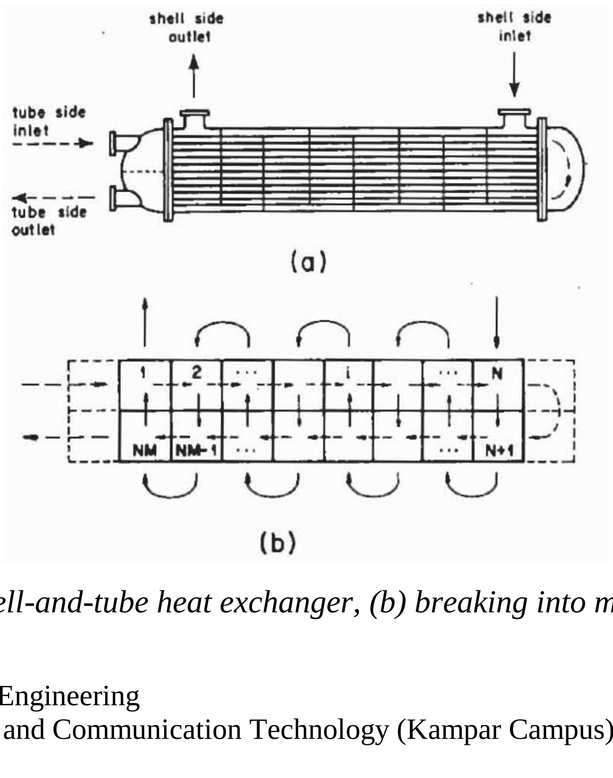


Figure 2.3.1: (a) Shell-and-tube heat exchanger, (b) breaking into multiple elements.

In the case of this project, M was set to 1 and N was set to 10 for convenience to form the calculation. Hence, a heat exchanger with 10 cells was considered. The following mathematical model of the shell-and-tube exchanger was proposed by Correa and Marchetti, which is in the form of algebraic equations.

First equation, the tube-side dimensionless temperature in cell *i* at time *k+1* is:

$$\phi_t^{k+1}(i) = a_1[\phi_s^{k}(i) + \phi_s^{k+1}(i)] + a_2[\phi_t^{k}(i-1) + \phi_t^{k+1}(i-1)] + a_3\phi_t^{k}(i)$$

Second equation, the shell-side dimensionless temperature in cell *j* at time *k+1* is:

$$\phi_s^{k+1}[L(j)] = b_1\{\phi_t^{k}[L(j)] + \phi_t^{k+1}[L(j)]\} + b_2\{\phi_s^{k}[L(j-1)] + \phi_s^{k+1}[L(i-1)]\} + b_3\phi_s^{k}[L(j)]$$

Where *t* denoted tube side, *k* denoted time, *i* denoted the cell numbers following a tube-side fluid trajectory, and *L(j)* denoted the vector of cell numbers following the shell-side fluid trajectory. The a_1, a_2, a_3, b_1, b_2, and b_3 are coefficients calculated based on the geometry and fluid properties of the heat exchanger. For the equations to calculate the coefficients a_1-a_3 and b_1-b_3, the interested reader is referred to reference (CORREA & MARCHETTI, 1987).

By using these algebraic equations, the dynamic model of shell-and-tube heat exchanger with finer details can be simulated. However, it needs to do some appropriate transformations so that these equations can be implemented in C#.

2.4 An Overview of Firebase

Firebase is a platform for mobile and web application developing. It was founded by Andrew Lee and James Tamplin in 2011. Initially, it was proposed to be an online chatting service. However, the plans were changed later. Firebase was acquired by Google in 2014 (Tamplin, 2014). More and more services are being introduced into Firebase to support many products. Firebase enables the developer to build more powerful and scalable applications by providing Firebase API for different platforms, such as iOS, Android, Web and so on. Developers can use the Firebase SDK to write their own software. In November 2016, Firebase announced to officially support for Unity game engine by offering its SDK (Kerpelman, 2016). Therefore, Unity game developer can take advantages of Firebase features.

One of the features of Firebase is the Firebase Real-time Database, which is a cloud-hosted NoSQL database (Firebase, n.d.). The developer can build applications without the need of servers since Firebase will provide it. Moreover, data are stored as a JSON file, where tabular relations are used rather than traditional relational databases. Any changes to the data will be stored and synchronized across all clients in real-time if the clients are online. For the offline client, the data are stored locally as caches so it makes the Firebase-powered applications responsive even though they go offline. The data is automatically synchronized once the client is reconnected to the server.

Due to the highly-responsive data synchronization with latency in the range of milliseconds, Firebase has been applied to the different field, especially time-critical field. For example, Firebase is confirmed to an appropriate correspondence arrangement that fulfilled the urgent nature of medical training (Alsalemi, et al., 2017). Firebase has also been used to develop an Android application for the purpose of rescuers in emergencies (Berbakov, et al., 2017).

CHAPTER 3 SYSTEM DESIGN

3.1 Design Specifications

3.1.1 The Specification of the Heat Exchanger

In this project, the heat exchanger model to be created is model HE104-4, which is a shell-and-tube heat exchanger unit manufactured by SOLTEQ. The company offers various specifications and details of the heat exchanger, so this information will be used as a reference for developing the simulator.

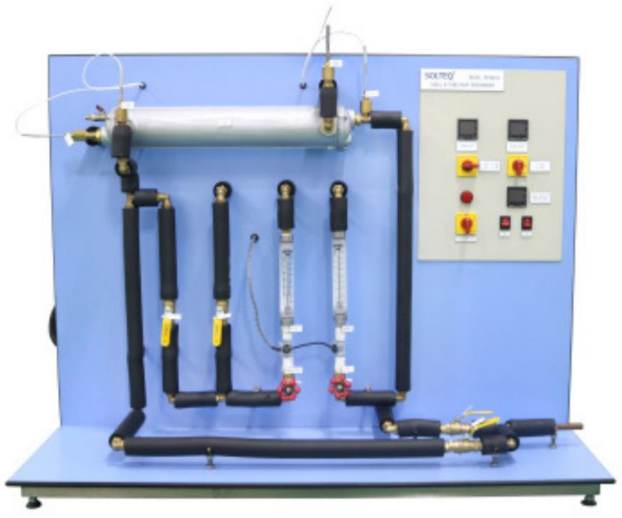


Figure 3.1.1: The physical unit of the HE104-4 shell-and-tube heat exchanger.

According to the company, 316 stainless steel is chosen as the material of both shell and tube, so the density is known as 7990 kg/m^3, and the thermal conductivity is 16.3 W/m-k. Other than that, the geometry information of the shell-and-tube heat exchanger is as follows:

- The inner diameter of tube = 2.56 mm = 0.00256 meter
- The outer diameter of tube = 3.20 mm = 0.0032 meter
- The length of tube = 508 mm = 0.508 meter
- The number of tubes = 55
- The inner diameter of shell = 34.80 mm = 0.0348 meter

In addition, the inside and outside tubes are equipped with 3 temperature sensors for accurate measurement of fluid temperature. The user of this equipment can adjust the flow rate and flow direction of the fluid by using a valve. When experimenting with such a heat exchanger, the equipment requires only cold water supply, and the hot water system is completely independent. A hot storage tank is equipped with an immersion type heater and an adjustable temperature controller which can maintain a temperature of around $\pm 1℃$. Based on this information, the main controllable parameters are obtained: the *fluid temperature*, the *fluid flow rate*, and the *flow direction of the fluid*.

The specifications of the above equipment (including geometric information and controllable parameters) are important in this project as they will be used as a reference for the mathematical model in this project.

3.1.2 Methodologies and General Work Procedures

The purpose of this project is to develop a heat exchanger simulator with real-time database. In order to develop this simulator, the Unity editor was selected as the development software in this project. This editor is used to create the interface of the simulator, and the Firebase is chosen to work with Unity editor. The Firebase acts as a back-end service, enabling the real-time database functionality in this project by providing services for storing and retrieving data in real time.

The development of this simulator is a team-based project, which includes another student, Kong Yee Kian. Both were responsible for different parts of the development, which will be described in the later sections.

3.1.3 Tools to Use

This project will be developed on the Windows 10 system platform. The following software and tools were installed for the development:

Operating System	• Windows 10
Main software	• Unity Editor
Integrated development environment (IDE)	• Microsoft Visual Studio 2017
Database	• Google Firebase
Software Development Kit (SDK)	• Firebase Unity SDK
Other	• Internet connection

Table 3.1.3: The tools used in the development.

3.2 Task Partitioning

3.2.1 Planning of the Task Partitioning

After discussion, the simulation program was decided to be primarily comprised of 3 scenes, namely the *Start Menu Scene*, the *Main Game Scene* as well as the *History Scene*. They are designed to be interconnected so that users can swap between these screens when clicking on a particular button. Their relationship is shown in figure 3.2.1.

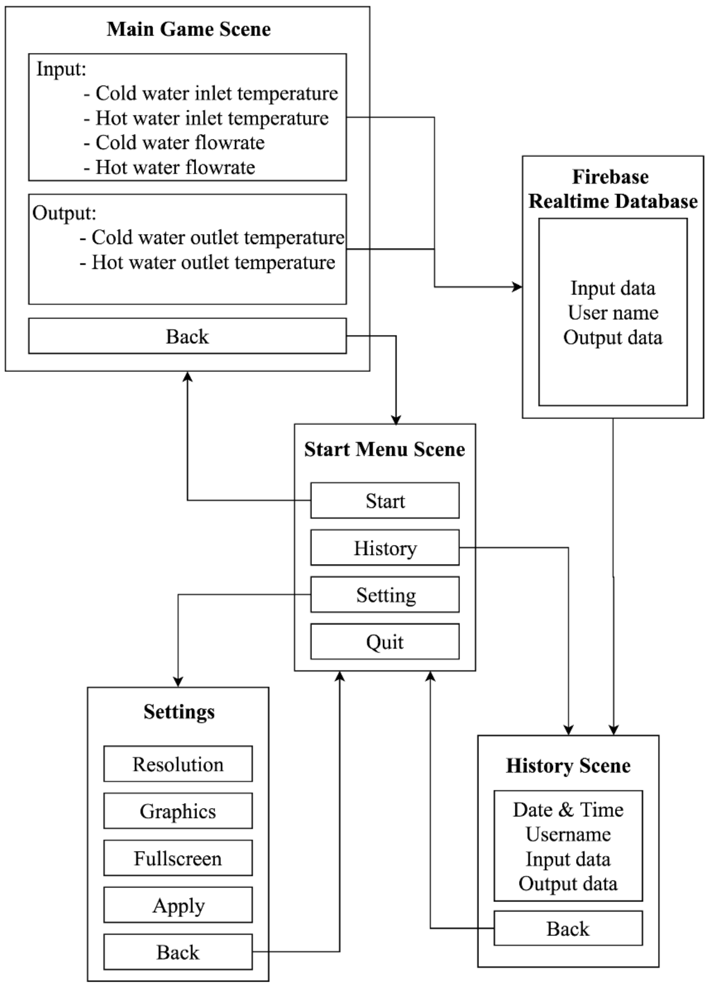


Figure 3.2.1: The relationship of scenes in the simulation program.

The assignment of tasks will be partitioned according to the figure above. One of the students, Tok Zhi Sung will be responsible for the *Start Menu Scene* and *History Scene*, while another student, Kong Yee Kian will be responsible for the *Main Game Scene* to create the interface of the simulation program. During the development process, all scenes require corresponding C# scripts to manipulate the data and control system, so they will work together to write the C# scripts for the systems. The C# scripts contain the algorithms for manipulating input data and computing output data according to the mathematical model of the heat exchanger in the main game scene.

3.2.2 The Requirements of This Project

The *Start Menu Scene* has 4 functions. The first function is to link with the main game scene created by another student, which will bring the user to do the simulation of a heat exchanger system. The second function is to link with the *History Scene*, which is a scene to show the past simulation results in tabular form. Moreover, the third function is to configure the graphics settings, such as resolution, quality of graphics and full screen. The last function is to exit the program. These features will be implemented by a script called “Menu.cs”.

When the user starts the simulation in the *Main Game Scene*, the simulation results are stored locally or in the cloud as long as there is data to be recorded. There are two types of data to be recorded, namely input data and output data. For input data, they come from the UI of the *Main Game Scene*, such as cold water inlet temperature, hot water inlet temperature, cold water flow rate and hot water flow rate. These data are then manipulated by an algorithm of the heat exchanger and then produce the output data. During the simulation, the input data and output data are passed immediately to the script named “DatabaseHandler.cs”. In short, this project is responsible for the communication between the *Main Game Scene* and the Firebase Realtime Database. The simulation data would be stored in the local storage as a text file to make the record available even without an internet connection. Moreover, this project also makes sure that the algorithm of the heat exchanger is integrated with the *Main Game Scene*.

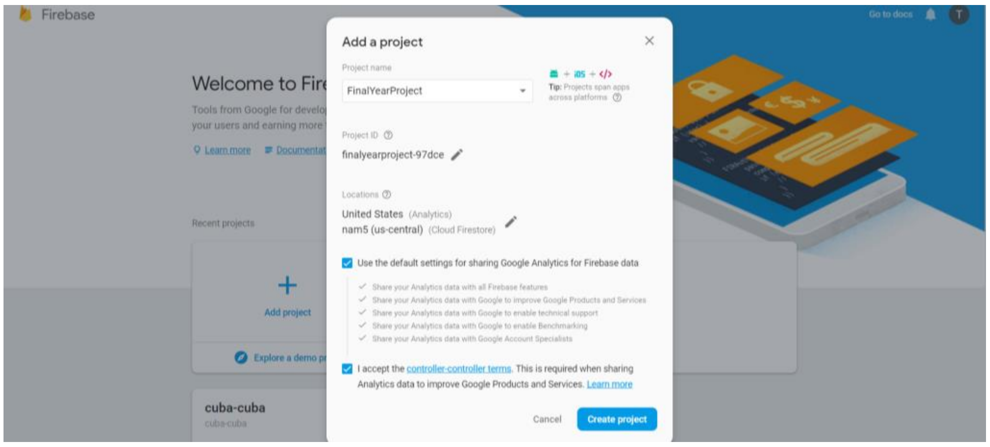
After the simulation, the user can check their simulation records in the *History Scene*, which is presented in a tabular form. Each row of the record is identified by a key that is a combination of the date and time when the simulation was performed. The user can click on the buttons on the side to examine the input data and output data in detail. Therefore, the main function of *History Scene* is to retrieve the past simulation data from Firebase Database. This function will be implemented by a script called “RetrieveHistory.cs”. In addition to this, another script called “HistoryObject.cs” was created in order to work with this script.

3.3 System Design

According to the task partition and the requirements mentioned above, the tasks in this project will be described in more detailed in the following section.

3.3.1 The Setup of the Firebase Realtime Database

Figures 3.3.1 (a) – (e) shows the steps performed to set up the Firebase Realtime Database. First of all, before adding Firebase to the Unity project, the user/developer (i.e., the student) would need to create a Firebase project to connect to the Unity project. The Google Firebase required a Google Account to be signed in. Once this is done, the new project was added in the Firebase console. For example, in this project, the project name was set as "FinalYearProject". After the required field was clicked to accept the terms and conditions, the project would be created.



Add a project

Project name: FinalYearProject

Tip: Projects span apps across platforms

Project ID: finalyearproject-97dce

Locations: United States (Analytics), nam5 (us-central) (Cloud Firestore)

[✓] Use the default settings for sharing Google Analytics for Firebase data
- ✓ Share your Analytics data with all Firebase features
- ✓ Share your Analytics data with Google to improve Google Products and Services
- ✓ Share your Analytics data with Google to enable technical support
- ✓ Share your Analytics data with Google to enable Benchmarking
- ✓ Share your Analytics data with Google Account Specialists

[✓] I accept the controller-controller terms. This is required when sharing Analytics data to improve Google Products and Services. Learn more

Figure 3.3.1(a): The project "FinalYearProject" was created in Firebase Console.

Firebase will automatically configure resources for user's Firebase project. After completing this process, the user was taken to the Firebase project overview page in the Firebase console. Then, the user is allowed to click on the Unity icon to register the Unity project with Firebase.

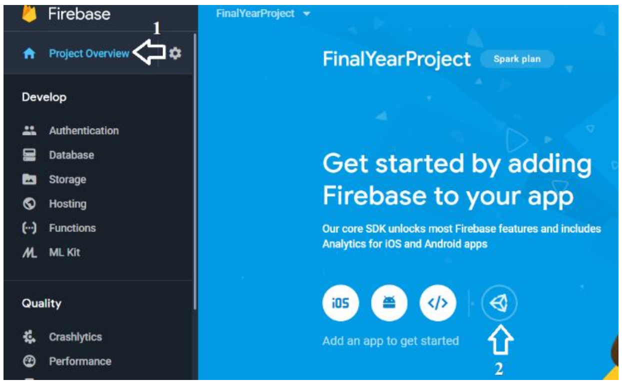
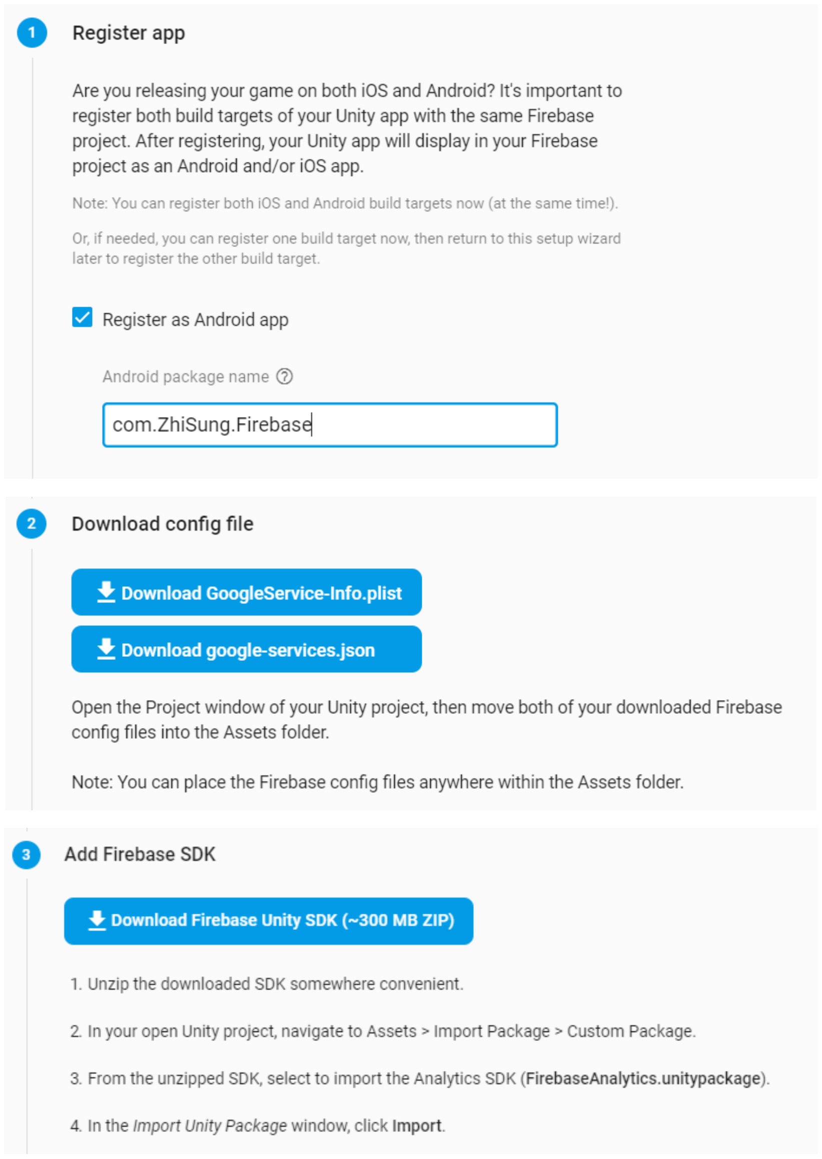

Firebase
1
Project Overview
Develop
Authentication
Database
Storage
Hosting
Functions
ML Kit
Quality
Crashlytics
Performance
FinalYearProject
FinalYearProject
Spark plan
Get started by adding Firebase to your app
Our core SDK unlocks most Firebase features and includes Analytics for iOS and Android apps
iOS
Add an app to get started
2

Figure 3.3.1(b): The Unity icon was clicked to launch the setup workflow.

In the next screen, Firebase will guide the user step by step to register his/her Unity project. Follow the instructions written by Firebase, the setup of Firebase was completed. The steps 1, 2 and 3 are important. The registered package name (format: com.CompanyName.UnityProductName) in step 1 was used in the Unity configuration. Besides that, the configuration files and the Firebase Unity SDK downloaded in step 2 and 3 would need to be imported into Unity editor later.



1 Register app

Are you releasing your game on both iOS and Android? It's important to register both build targets of your Unity app with the same Firebase project. After registering, your Unity app will display in your Firebase project as an Android and/or iOS app.

Note: You can register both iOS and Android build targets now (at the same time!).

Or, if needed, you can register one build target now, then return to this setup wizard later to register the other build target.

[✓] Register as Android app

Android package name: com.ZhiSung.Firebase

2 Download config file

Download GoogleService-Info.plist
Download google-services.json

Open the Project window of your Unity project, then move both of your downloaded Firebase config files into the Assets folder.

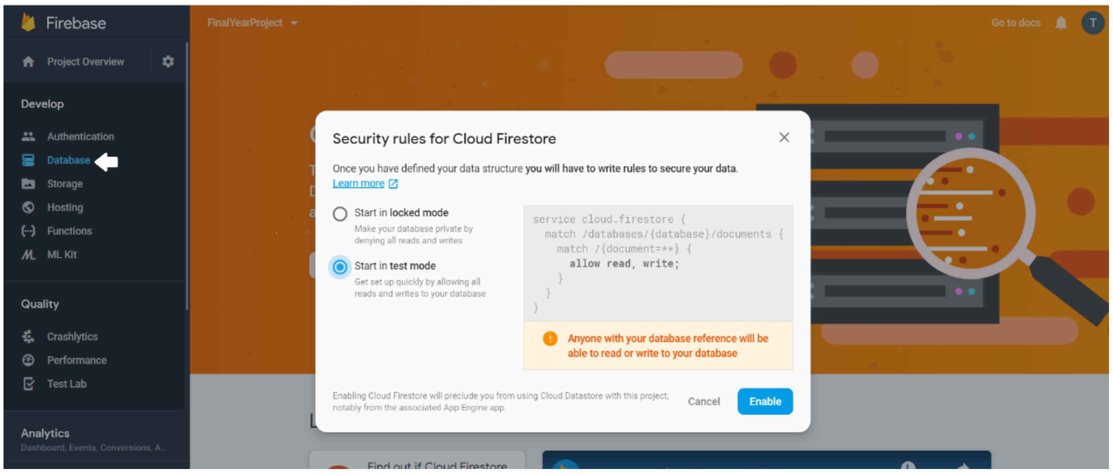
Note: You can place the Firebase config files anywhere within the Assets folder.

3 Add Firebase SDK

1. Unzip the downloaded SDK somewhere convenient.
2. In your open Unity project, navigate to Assets > Import Package > Custom Package.
3. From the unzipped SDK, select to import the Analytics SDK (**FirebaseAnalytics.unitypackage**).
4. In the *Import Unity Package* window, click **Import**.

Figure 3.3.1(c): The Unity project was registered with Firebase in step 1, the configuration files and the Firebase SDK were downloaded in the steps 2 and 3.

After register the project with Firebase, the user is allowed to click on the database at the left side's panel, where it will bring the user to set up the database. In this project, the test mode was selected.



Security rules for Cloud Firestore

Once you have defined your data structure you will have to write rules to secure your data.
Learn more

Start in locked mode
Make your database private by denying all reads and writes

[✓ Start in test mode]
Get set up quickly by allowing all reads and writes to your database

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write;
    }
  }
}
```

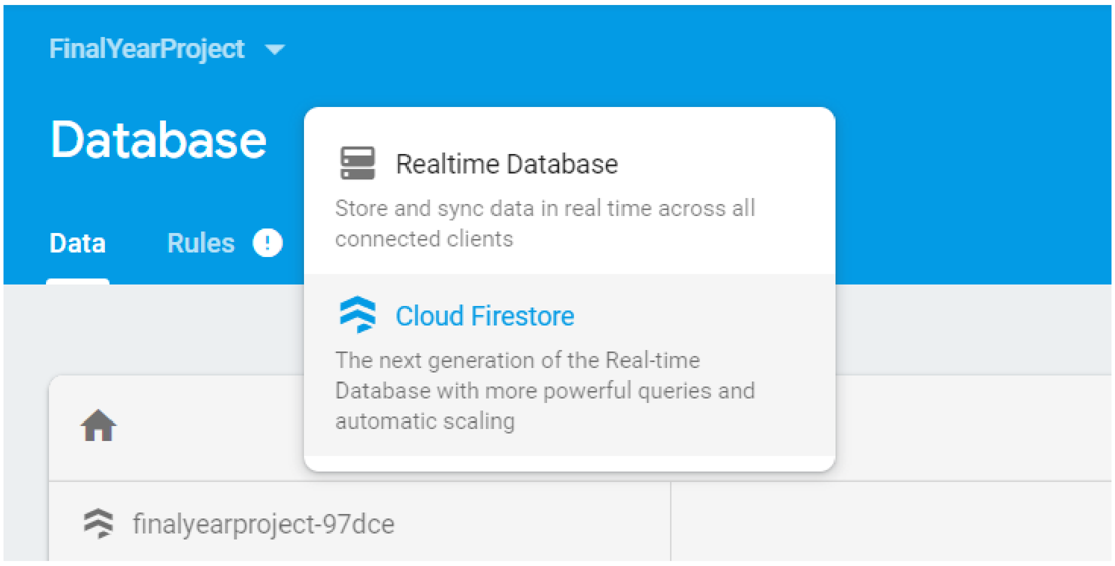
Anyone with your database reference will be able to read or write to your database

Enabling Cloud Firestore will preclude you from using Cloud Datastore with this project, notably from the associated App Engine app.

Cancel Enable

Figure 3.3.1(d): The "test mode" was selected to setup database.

The default database is Cloud Firestore. However, this project switched it to the Realtime Database as shown in Figure 3.3.1(e).



Realtime Database
Store and sync data in real time across all connected clients

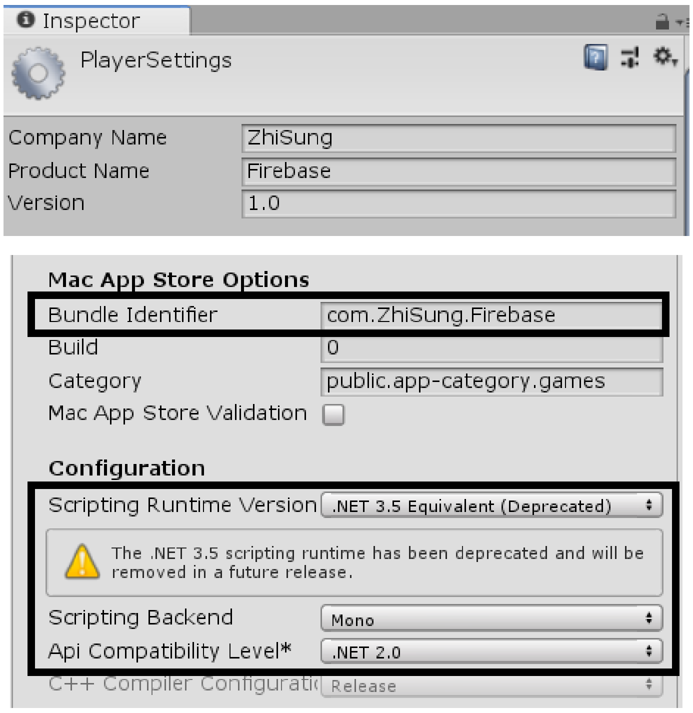
Cloud Firestore
The next generation of the Real-time Database with more powerful queries and automatic scaling

Figure 3.3.1 (e): The default database was changed to the Realtime Database.

Until this step, the setup of Firebase's Realtime Database was completed.

3.3.2 Unity Editor Setup

Figures 3.3.2 (a) – (c) shows the steps performed to set up the Unity Editor. First of all, the Unity Editor was started and a new 3D project was created. Once the project was opened in the Unity Editor, the “Player Settings” was opened by clicking on “File” then “Build Settings”. The “Player Settings” was prompted in the “Inspector” window. In this window, the “Company Name” field and the “Product Name” field were changed to the name registered in Firebase project, including the “Bundle Identifier”. At the configuration part, the “Scripting Runtime Version” was set to “.Net 3.5 Equivalent”, the “Scripting Backend” was set to “Mono”, and the “Api Compatibility Level” was set to “.Net 2.0”.


Inspector
PlayerSettings

Company Name	ZhiSung
Product Name	Firebase
Version	1.0

Mac App Store Options

Bundle Identifier	com.ZhiSung.Firebase
Build	0
Category	public.app-category.games
Mac App Store Validation	☐

Configuration

Scripting Runtime Version	.NET 3.5 Equivalent (Deprecated)

The .NET 3.5 scripting runtime has been deprecated and will be removed in a future release.

Scripting Backend	Mono
Api Compatibility Level*	.NET 2.0
C++ Compiler Configurati	Release

Figure 3.3.2(a): The player settings were configured in the Unity editor.

After that, the “Assets” folder was founded in the Project Window. The configuration files downloaded from Firebase was moved into the “Assets” folder.

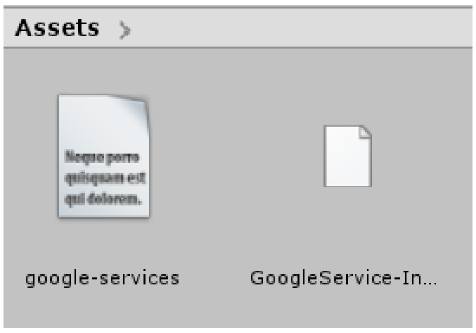
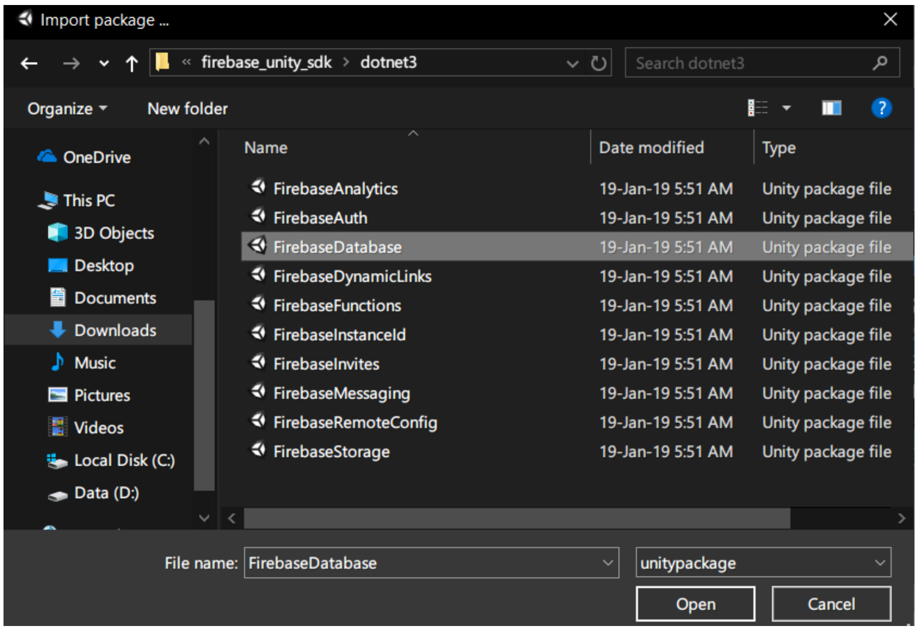


Figure 3.3.2(b): The Firebase configuration files were moved to the Unity project.

Next, the Firebase Unity SDK downloaded from Firebase console was unzipped. The unzipped SDK have will have 2 folders: “dotnet3” and “dotnet4”, each of them contains different Unity package files. To import this package, the following steps were performed: go back to Unity editor, right click on the “Assets” folder, then click “Import Package” > “Custom Package”, locates the Unity package file which called “FirebaseDatabase” in “dotnet3”.



Name	Date modified	Type
FirebaseAnalytics	19-Jan-19 5:51 AM	Unity package file
FirebaseAuth	19-Jan-19 5:51 AM	Unity package file
FirebaseDatabase	19-Jan-19 5:51 AM	Unity package file
FirebaseDynamicLinks	19-Jan-19 5:51 AM	Unity package file
FirebaseFunctions	19-Jan-19 5:51 AM	Unity package file
FirebaseInstanceId	19-Jan-19 5:51 AM	Unity package file
FirebaseInvites	19-Jan-19 5:51 AM	Unity package file
FirebaseMessaging	19-Jan-19 5:51 AM	Unity package file
FirebaseRemoteConfig	19-Jan-19 5:51 AM	Unity package file
FirebaseStorage	19-Jan-19 5:51 AM	Unity package file

File name: FirebaseDatabase
unitypackage

Figure 3.3.2(c): The Firebase Database SDK was unzipped and the “FirebaseDatabase” was imported to the Unity project.

3.3.3 Scenes Creation

Once the setup of the Firebase Realtime Database and Unity Project was completed, the next step was to create the scene. A folder called “Scenes” was created in the “Assets” folder to store all the scenes created in this project. After that, 2 new scenes called “Start Menu” and “History” were created in this folder.

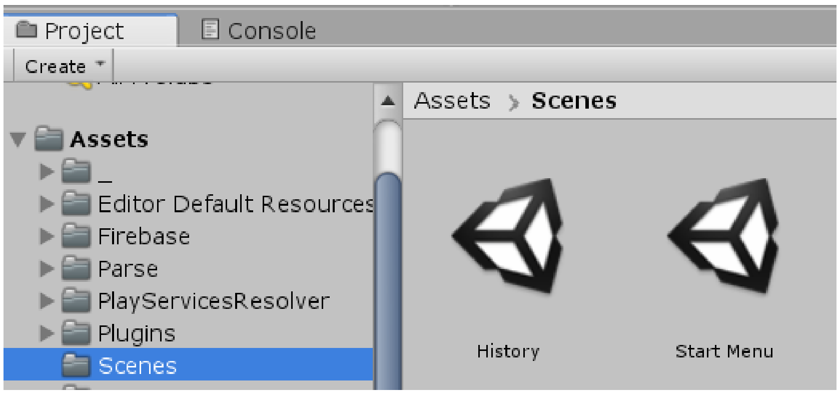


Figure 3.3.3: A folder named “Scenes” was created to store the scene objects.

3.3.3.1 Start Menu Scene

Since the *Start Menu Scene* is a 2D interface, so the first thing to do was to set this scene to 2D. Under the scene tab, the 2D button was clicked.

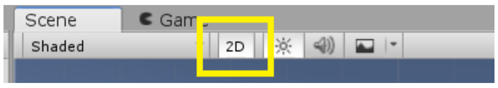
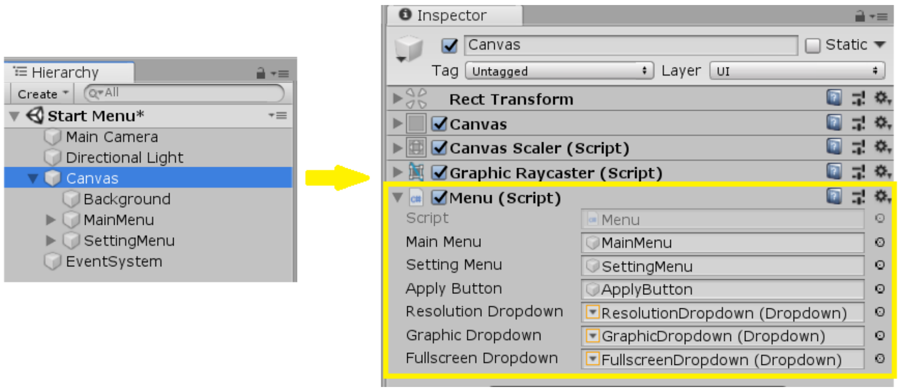


Figure 3.3.3.1(a): The scene was set to a 2D interface.

The succeeding steps are to create UI elements. In Unity, "Canvas" is known as a *GameObject* with a *Canvas Component* on it, and all of the UI elements should be created inside of the Canvas. Hence, a Canvas object was created in the Hierarchy Window. Inside of the Canvas object, it contained 3 *GameObjects*: "Background", "MainMenu", and "SettingMenu". The reason to create these objects in this logical hierarchy was to facilitate understanding of developers and development. Besides that, a custom C# script named "Menu.cs" was added as one of the components of Canvas.



Start Menu*
- Main Camera
- Directional Light
- Canvas
  - Background
  - MainMenu
  - SettingMenu
- EventSystem

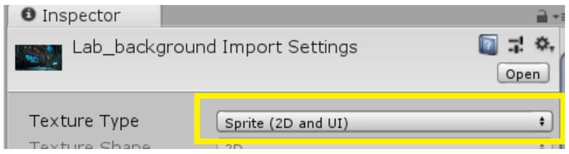
Canvas
Tag: Untagged Layer: UI

Canvas
Canvas Scaler (Script)
Graphic Raycaster (Script)
Menu (Script)

Script	Menu
Main Menu	MainMenu
Setting Menu	SettingMenu
Apply Button	ApplyButton
Resolution Dropdown	ResolutionDropdown (Dropdown)
Graphic Dropdown	GraphicDropdown (Dropdown)
Fullscreen Dropdown	FullscreenDropdown (Dropdown)

Figure 3.3.3.1(b): The children of Canvas (left) and the C# script named "Menu" was added as a component (right).

The "Background" object was a "Panel", which is a *GameObject* which an *Image Component* on it. In order to set the source image for this component, a JPEG/PNG image was added to the "Assets" folder. This image could not be used directly since the texture type should be configured to "Sprite (2D and UI)". Once the image's *texture type* was determined as *sprite*, it could be used as the source image. So, it was dragged and dropped to the "source image" field in Inspector Window of "Background" object.

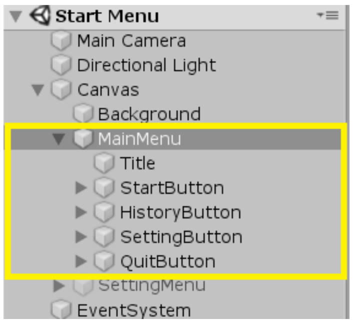


Lab_background Import Settings

Texture Type: Sprite (2D and UI)

Figure 3.3.3.1(c): The image's texture type was changed to Sprite.

Basically, the "MainMenu" object was an empty *GameObject*, it acted as a container to store other *GameObjects*. Inside of the "MainMenu", it contained 5 *GameObjects* (children) as shown in Figure 3.3.3.1(d).

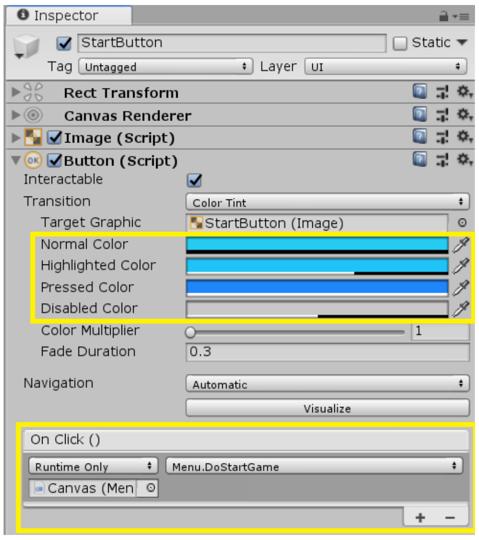


```
▼ Start Menu
    Main Camera
    Directional Light
  ▼ Canvas
      Background
    ▼ MainMenu
        Title
      ▶ StartButton
      ▶ HistoryButton
      ▶ SettingButton
      ▶ QuitButton
    ▶ SettingMenu
    EventSystem
```

Figure 3.3.3.1(d): The hierarchy of GameObjects in MainMenu.

The "Title" object was a "TextMeshPro - Text", which is a free development asset downloaded from Unity Asset Store. It is the perfect replacement for Unity's built-in UI text, providing substantial visual quality improvements while providing users with incredible flexibility in text styles and textures. In the Inspector Window of this object, the *text* was changed to "HE-104 HEAT EXCHANGER SIMULATOR".

The remaining *GameObjects* in the "MainMenu" were Unity's built-in UI *GameObjects* - "Button(s)". In the Inspector Window of "StartButton" (Figure 3.3.3.1(e)), the *normal colour*, *highlighted colour*, and *pressed colour* field was set with different opacity to segregate and indicate the normal status, mouse-hover status, and mouse-click status of button respectively. The crucial part was to add a *listener* to this button. In Unity, when a button is pressed, the registered *listeners* of onClick will be performed. For example, the method "Menu.DoStartGame()" was set as the listener of "StartButton", which swapped the current scene to the *Main Game Scene*.



```
Inspector
☑ StartButton                         ☐ Static
Tag  Untagged            Layer  UI
▶ Rect Transform
▶ Canvas Renderer
▶ ☑ Image (Script)
▼ ☑ Button (Script)
  Interactable        ☑
  Transition          Color Tint
    Target Graphic    StartButton (Image)
    Normal Color
    Highlighted Color
    Pressed Color
    Disabled Color
    Color Multiplier  1
    Fade Duration     0.3
  Navigation          Automatic
                      Visualize
  On Click ()
  Runtime Only        Menu.DoStartGame
  Canvas (Men
                                        + −
```

Figure 3.3.3.1(e): The field of "StartButton" that needed to be configured.

Similarly, the *normal colour*, *highlighted colour*, and *pressed colour* field of "HistoryButton", "SettingButton", and "QuitButton" were set with different opacity. However, the listener of "HistoryButton" was "Menu.DoOpenHistory()", which swapped the current scene to *History Scene*. The listener of "SettingButton" was "Menu.DoOpenSetting()", which hides all of the *GameObjects* in "MainScene" and showed the "SettingMenu". In addition, the listener of "QuitButton" was "Menu.DoQuitGame()" which would terminate the application. Finally, the "MainMenu" shall look like Figure 3.3.3.1(f).

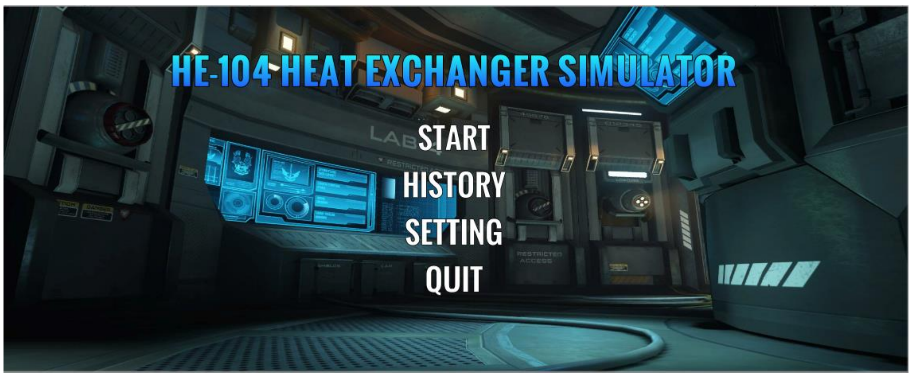
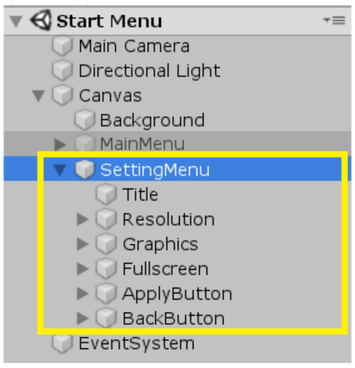


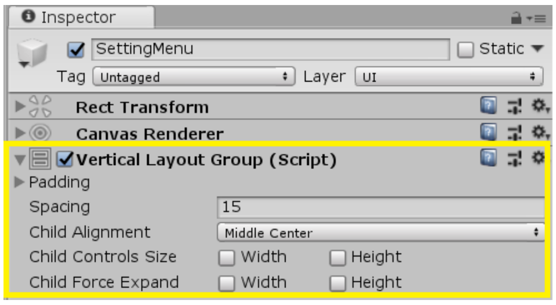
Figure 3.3.3.1(f): The screenshot of "MainMenu".

Proceeding to the next *GameObject* - "SettingMenu", which was a container to store 6 *GameObjects* (children) as shown in Figure 3.3.3.1(g). In order to align its children vertically, a component called "Vertical Layout Group" was attached to it.



```
▼ Start Menu
    Main Camera
    Directional Light
  ▼ Canvas
      Background
    ▶ MainMenu
    ▼ SettingMenu
        Title
      ▶ Resolution
      ▶ Graphics
      ▶ Fullscreen
      ▶ ApplyButton
      ▶ BackButton
    EventSystem
```

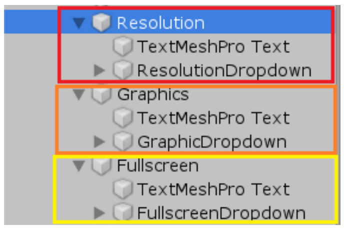
Figure 3.3.3.1(g): The hierarchy of children in "SettingMenu".



Inspector
SettingMenu ☐ Static
Tag Untagged Layer UI
▶ Rect Transform
▶ Canvas Renderer
▼ ☑ Vertical Layout Group (Script)
▶ Padding
Spacing 15
Child Alignment Middle Center
Child Controls Size ☐ Width ☐ Height
Child Force Expand ☐ Width ☐ Height

Figure 3.3.3.1(h): "Vertical Layout Group" component was added to "SettingMenu".

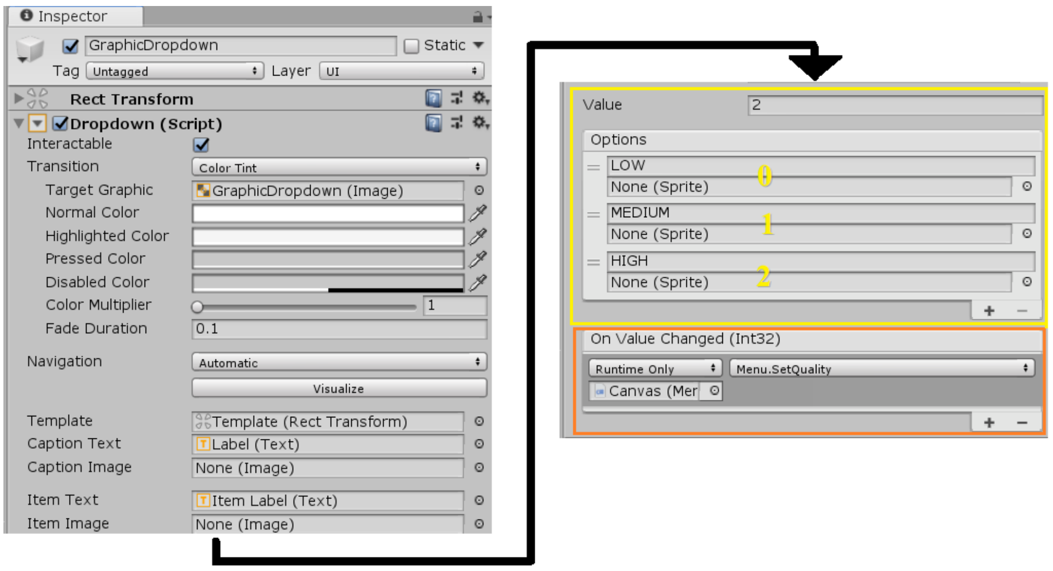
The first child “Title” of “SettingMenu” was an object of “TextMeshPro - Text”. Its *text* field was set as “SETTINGS”. The second, third and fourth children of “SettingMenu” were similar. Each of them had a “TextMeshPro - Text” object and a “Dropdown” object as shown in Figure 3.3.3.1(i). The *text* field of “TextMeshPro Text” was set as “Resolution”, “Graphics”, and “Fullscreen” respectively.



```
▼ Resolution
    TextMeshPro Text
  ▶ ResolutionDropdown
▼ Graphics
    TextMeshPro Text
  ▶ GraphicDropdown
▼ Fullscreen
    TextMeshPro Text
  ▶ FullscreenDropdown
```

Figure 3.3.3.1(i): The children of “Resolution”, “Graphics” and “Fullscreen”.

In Unity, “Dropdown” is a *GameObject* that presents a list of options when clicked. When an option is chosen, a dropdown event occurs a callback is sent to the registered listeners of onValueChanged. Refer to Figure 3.3.3.1(j)., “Low”, “Medium”, and “High” were added manually as the options of “GraphicsDropdown”. Each option has value “0”, “1”, and “2” respectively. When an option was chosen, the value was passed to the registered listener “Menu.SetQuality” in order to change the graphics quality. In addition, the options of “ResolutionDropdown” were added automatically by the algorithms of “Menu.cs”. It added all of the resolutions supported by the monitor. The listener of “ResolutionDropdown” was “Menu.SetResolution”. For “FullscreenDropdown”, “DISABLE” and “ENABLE” were added as the options. Its listener was “Menu.SetFullscreen”, which exchanges the simulation program between windowed mode and fullscreen mode.



```
Inspector
✓ GraphicDropdown                    Static
Tag  Untagged          Layer  UI
Rect Transform
✓ Dropdown (Script)
Interactable        ✓
Transition          Color Tint
  Target Graphic    GraphicDropdown (Image)
  Normal Color
  Highlighted Color
  Pressed Color
  Disabled Color
  Color Multiplier  1
  Fade Duration     0.1
Navigation          Automatic
                    Visualize
Template            Template (Rect Transform)
Caption Text        Label (Text)
Caption Image       None (Image)
Item Text           Item Label (Text)
Item Image          None (Image)

Value               2
Options
  LOW
  None (Sprite)       0
  MEDIUM
  None (Sprite)       1
  HIGH
  None (Sprite)       2
On Value Changed (Int32)
  Runtime Only        Menu.SetQuality
  Canvas (Mer
```

Figure 3.3.3.1(j): The Inspector Window of “GraphicsDropdown”.

The last two children of "SettingMenu" were "ApplyButton" and "BackButton". "ApplyButton" was used to implement the changes of settings and keep the user on the settings page, while "BackButton" was used to bring the user back to the main menu page. Finally, the "SettingMenu" shall look like Figure 3.3.3.1(k).

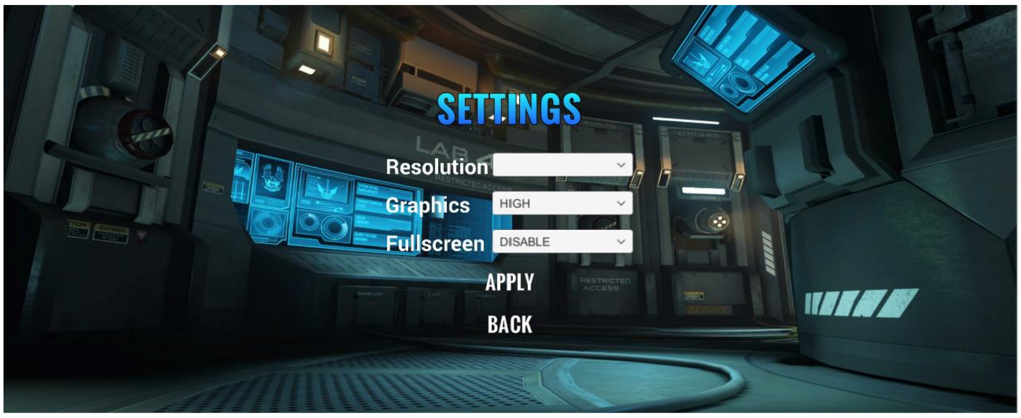


Figure 3.3.3.1(k): The screenshot of the settings menu.

3.3.3.2 History Scene

The *History Scene* was also a 2D interface. It displayed the past simulation results in the form of table. Although the table is a common UI object, the Unity editor does not provide such built-in UI objects. Hence, it must be created manually. First of all, the UI elements (*GameObjects*) were created as shown in Figure 3.3.3.2(a).

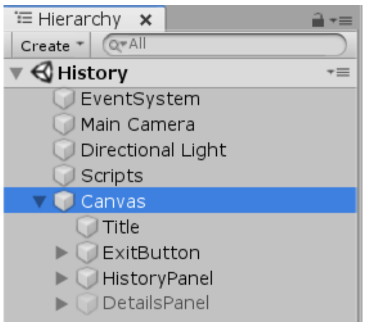
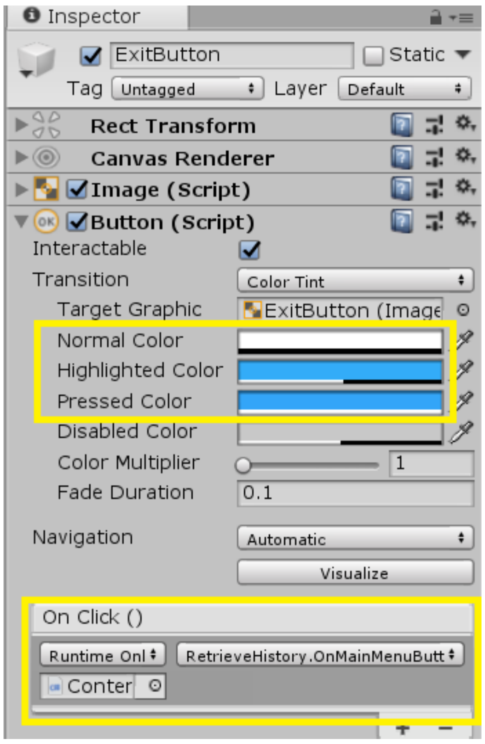


Figure 3.3.3.2(a): The UI elements in Canvas.

The first child of Canvas was "Title", which was an object of "TextMeshPro - Text". The *text* field was set to "HISTORY". The second child of Canvas was "ExitButton", which was a *button* object that brings the user back to the *Start Menu Scene* when clicked. Similarly, 3 normal colour, highlighted colour and pressed colour were set with different opacity. When user clicks it, the registered listener

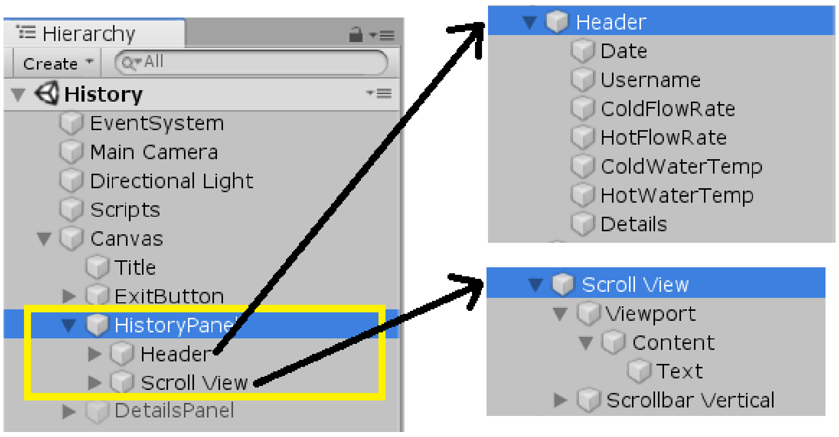
"RetrieveHistory.OnMainMenuButtonClick()" will be triggered (as shown in Figure 3.3.3.2(b)).



```
Inspector
ExitButton                         Static
Tag  Untagged        Layer  Default
Rect Transform
Canvas Renderer
Image (Script)
Button (Script)
Interactable          ✓
Transition            Color Tint
  Target Graphic      ExitButton (Image
  Normal Color
  Highlighted Color
  Pressed Color
  Disabled Color
  Color Multiplier                    1
  Fade Duration       0.1
Navigation            Automatic
                      Visualize
On Click ()
Runtime Onl    RetrieveHistory.OnMainMenuButt
Conter
```

Figure 3.3.3.2(b): The properties of "ExitButton" to be set.

In the "HistoryPanel", it contained 2 *GameObjects* namely "Header" and "Scroll View" as shown in Figure 3.3.3.2(c). For "Header", 7 children of type UI "Text" were added as the children. For "Scroll View", it was a built-in UI element. In Unity, "Scroll View" is used when a content that takes up a lot of space needs to be displayed in a small area. It provides functionality to scroll over this content. In addition, the content to be displayed was added as the children of "Content" in this hierarchy. Note that in order to function properly, "Content Size Fitter" was added in a later step.



```
Hierarchy
Create   All
History
  EventSystem
  Main Camera
  Directional Light
  Scripts
  Canvas
    Title
    ExitButton
    HistoryPanel
      Header
      Scroll View
    DetailsPanel

Header
  Date
  Username
  ColdFlowRate
  HotFlowRate
  ColdWaterTemp
  HotWaterTemp
  Details

Scroll View
  Viewport
    Content
      Text
  Scrollbar Vertical
```

Figure 3.3.3.2(c): The children of "HistoryPanel".

Until this step, the “HistoryPanel” shall look like the figure below.

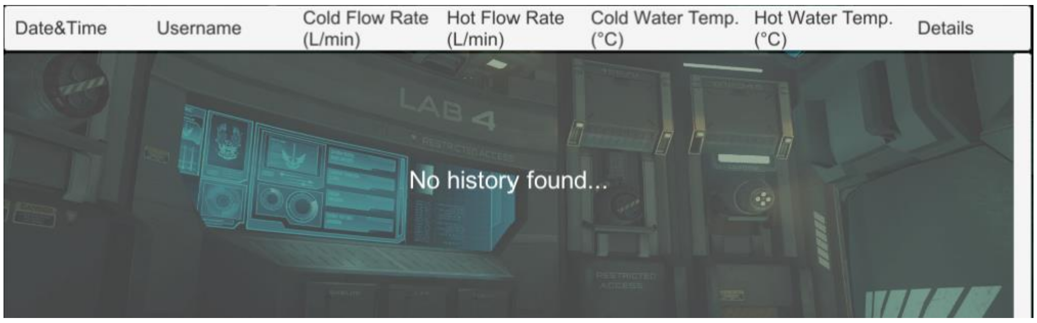


Figure 3.3.3.2(d): The screenshot of “HistoryPanel”.

No history was displayed in the “HistoryPanel” because the children of “Content” had not been set. At here, Unity’s Prefab system was used. Unity’s Prefab system allows the developer to create, configure, and store a *GameObject* complete with all its components, property values, and child *GameObjects* as a *reusable asset*. In other words, the “Prefab” asset acts as a template from which developer can create new “Prefab” instances in the Scene. As a summary, Prefab can help the developer create any number of *GameObjects* by using scripts.

Since the content to be displayed was similar to the “Header” object, so the Prefab asset could be created by modifying the “Header” object. The steps were:

1. In Hierarchy Window, duplicate the “Header” object.
2. Change the name of this duplicated object from “Header(1)” to “Prefab_List”.
3. Change the “Details” child to a *Button*.
4. Drag “Prefab_List” to the Assets folder in Project Window.

   *The button object’s listener was added in the C# script.

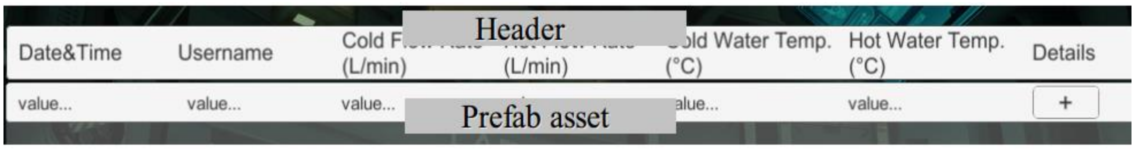


Figure 3.3.3.2(e): The “Header” object and “Prefab_list” asset had similar children.

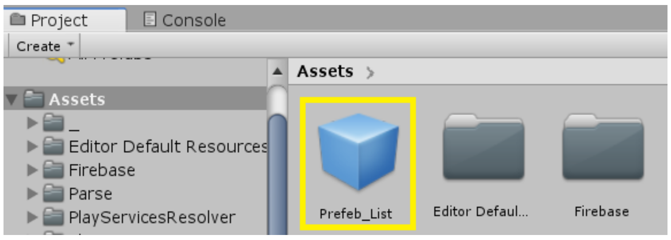


Figure 3.3.3.2(f): The “Prefab_List” asset was successfully created.

After creating the “Prefab_List” asset, 3 components were added to the “Content”. Refer to Figure 3.3.3.2(g), the component “Vertical Layout Group” was used to align its children vertically within the “Content” area. Next, the C# script “RetrieveHistory.cs” was added a component that control the behaviours of *GameObjects*. Then, all of the related *GameObjects* created in the Canvas were linked with this script. Last, a component called “Content Size Fitter” was added. It was important to make “Scroll View” function properly. The properties of these components were configured according to Figure 3.3.3.2(g).

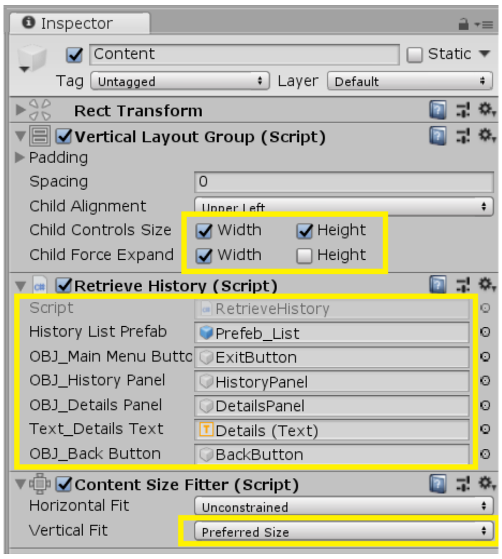

```
Inspector
Content                              Static
Tag Untagged            Layer Default
Rect Transform
Vertical Layout Group (Script)
Padding
Spacing                0
Child Alignment        Upper Left
Child Controls Size    [✓] Width   [✓] Height
Child Force Expand     [✓] Width   [ ] Height
Retrieve History (Script)
Script                 RetrieveHistory
History List Prefab    Prefeb_List
OBJ_Main Menu Buttc    ExitButton
OBJ_History Panel      HistoryPanel
OBJ_Details Panel      DetailsPanel
Text_Details Text      Details (Text)
OBJ_Back Button        BackButton
Content Size Fitter (Script)
Horizontal Fit         Unconstrained
Vertical Fit           Preferred Size
```

Figure 3.3.3.2(g): The components that newly added to “Content”.

The creation of the last child of Canvas “DetailsPanel” was similar to the “HistoryPanel”. The differences were, the child of “Header” was changed to a UI *button* named “BackButton”, and the child of “Content” was changed to a UI Text named “Details”.

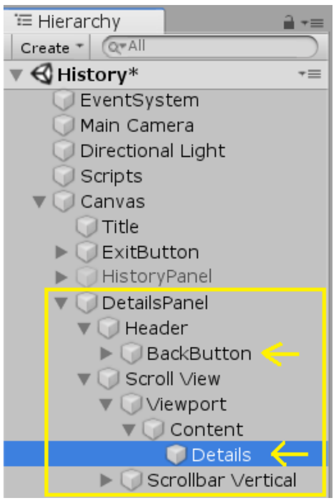

```
Hierarchy
Create   All
History*
  EventSystem
  Main Camera
  Directional Light
  Scripts
  Canvas
    Title
    ExitButton
    HistoryPanel
    DetailsPanel
      Header
        BackButton
      Scroll View
        Viewport
          Content
            Details
        Scrollbar Vertical
```

Figure 3.3.3.2(h): The different parts of “DetailsPanel”.

In the end, the history scene should look like the figure below.

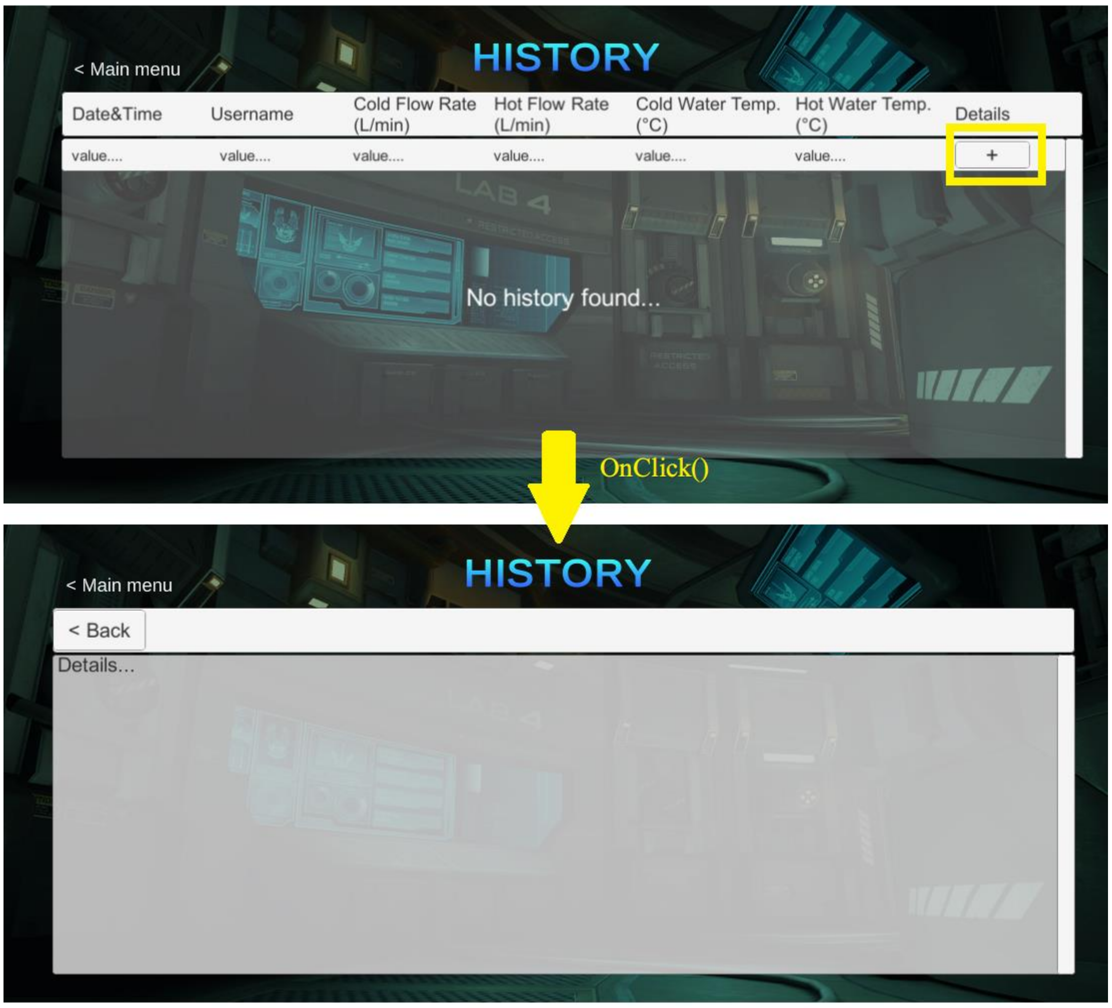


Figure 3.3.3.2(i): The screenshot of the History Scene.

3.3.4 <u>The Explanation of the C# Scripts</u>

In Unity, the behaviour of *GameObjects* is controlled by the *components* that are attached to them. Unity enables the developer to create own components utilizing scripts so that it can trigger game events, modify the properties of a component over time and react to the user input. In this section, the C# scripts created in this project will be explained.

3.3.4.1 The “Menu” Script

In the *Start Menu Scene*, “Menu.cs” was attached to the Canvas as one of the components. Noted that every Unity script is derived from the base class – “MonoBehaviour”. So, “Menu.cs” must implement this class (interface) explicitly.

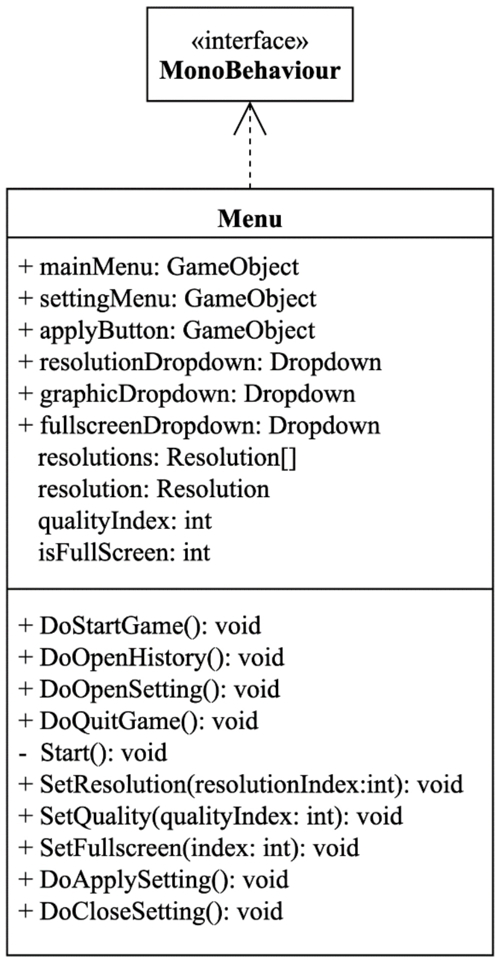

«interface»
MonoBehaviour
Menu
+ mainMenu: GameObject
+ settingMenu: GameObject
+ applyButton: GameObject
+ resolutionDropdown: Dropdown
+ graphicDropdown: Dropdown
+ fullscreenDropdown: Dropdown
resolutions: Resolution[]
resolution: Resolution
qualityIndex: int
isFullScreen: int
+ DoStartGame(): void
+ DoOpenHistory(): void
+ DoOpenSetting(): void
+ DoQuitGame(): void
- Start(): void
+ SetResolution(resolutionIndex:int): void
+ SetQuality(qualityIndex: int): void
+ SetFullscreen(index: int): void
+ DoApplySetting(): void
+ DoCloseSetting(): void

Figure 3.3.4.1: The UML notation of “Menu” script.

Explanation of methods in “Menu”:

1. DoStartGame()
   This method is triggered when the user clicks “StartButton”, which changes the current scene to the main menu scene by Scene Manager.
2. DoOpenHistory()
   This method is triggered when the user clicks “HistoryButton”, which changes the current scene to the history scene by Scene Manager.

3. DoOpenSetting()

   This method is triggered when the user clicks “SettingButton”, which hides the GameObject “mainMenu”, shows “settingMenu”, and load the current graphics settings.
4. DoQuitGame()

   This method is triggered when the user clicks “QuitButton”, which terminates the program.
5. Start()

   This method is called automatically when the script is enabled. It is called only once in the lifetime of the script. Hence, this method is used to initialize the properties in Menu. It also checks all the resolutions supported by the monitor and then adds them as the options of object “ResolutionDropdown”.
6. SetResolution(int resolutionIndex)

   This method is triggered when the user selects one of the options of “ResolutionDropdown”, which changes the value of local variable “resolution” to the selected resolution.
7. SetQuality(int qualityIndex)

   This method is triggered when the user selects one of the options of “GraphicDropdown”, which changes the value of local variable “qualityIndex” to the selected quality.
8. SetFullscreen(int index)

   This method is triggered when the user clicks “ResolutionDropdown”, which changes the value of local variable “isFullScreen” between fullscreen or windowed mode.
9. DoApplySetting()

   This method is triggered when the user clicks “ApplyButton”, which applies all the changes of setting.
10. DoCloseSetting()

    This method is triggered when the user clicks “BackButton”, which changes the current scene to the main menu scene by Scene Manager.

3.3.4.2 The "HistoryObject" Script

In *History Scene*, two scripts were created. "RetrieveHistory.cs" was attached to the "Content" object (child of "HistoryPanel"), and "HistoryObject.cs" was attached to "Scripts". For "HistoryObject.cs", it is a class with 2 private fields and few methods. The special thing about this class is that "Dictionary" is used to declare the private field "history". In C# programming, the "Dictionary<T Key, T Value>" represents a collection of key-value pairs of data. Developers can easily get the corresponding value of data by using a particular key. Therefore, "Dictionary" class provides functionality that works like a normal dictionary, using words (keys) to find meanings (values). In the case of this project, the "Dictionary" was cascaded in order to find a value by a combination of key (identifier and input type).

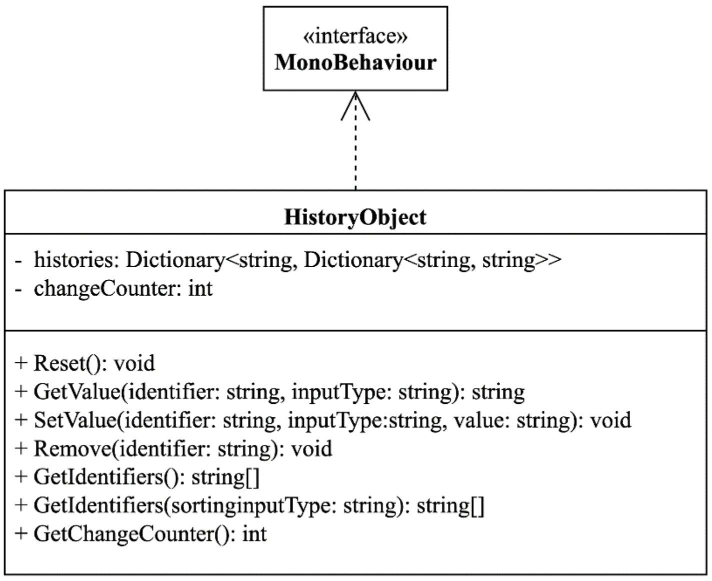

«interface»
MonoBehaviour
HistoryObject
- histories: Dictionary<string, Dictionary<string, string>>
- changeCounter: int
+ Reset(): void
+ GetValue(identifier: string, inputType: string): string
+ SetValue(identifier: string, inputType:string, value: string): void
+ Remove(identifier: string): void
+ GetIdentifiers(): string[]
+ GetIdentifiers(sortinginputType: string): string[]
+ GetChangeCounter(): int

Figure 3.3.4.2: The UML notation of the "HistoryObject" script.

Explanation of the methods:

1. Reset()
   This method clears the recorded keys and value pairs in the collection of "histories".
2. GetValue(string identifier, string inputType)
   This method finds the keys equal to "identifier" and "inputType" from the collection of "histories". It returns the value of if the key is found. If the key does not exist then return null.

3. SetValue(string identifier, string inputType, string value)

   This method adds key-value pairs to the collection of “histories”.
4. Remove(string identifier)

   This method removes key-value pairs from the collection of histories.
5. GetIdentifiers()

   This method returns all recorded keys through a string array.
6. GetIdentifiers(string sortinginputType)

   This method returns all recorded keys through a string array sorted by order by “sortinginputType”.
7. GetChangeCounter()

   This method returns the value of “changeCounter”. Once there are changes to the collection of “histories”, the “changeCounter” is incremented by one.

3.3.4.3 The “RetrieveHistory” Script

The “RetrieveHistory” was another script used in *History Scene*. The class “HistoryObject” created in the previous section was used to declare an object named “historyObject”. This script was attached to the “Content” (child of “HistoryPanel”).

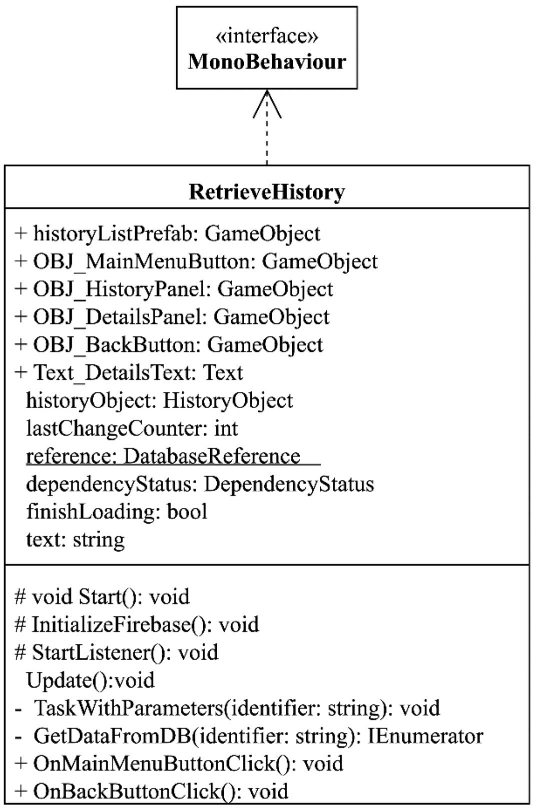

«interface»
MonoBehaviour
RetrieveHistory
+ historyListPrefab: GameObject
+ OBJ_MainMenuButton: GameObject
+ OBJ_HistoryPanel: GameObject
+ OBJ_DetailsPanel: GameObject
+ OBJ_BackButton: GameObject
+ Text_DetailsText: Text
historyObject: HistoryObject
lastChangeCounter: int
reference: DatabaseReference
dependencyStatus: DependencyStatus
finishLoading: bool
text: string
# void Start(): void
# InitializeFirebase(): void
# StartListener(): void
Update():void
- TaskWithParameters(identifier: string): void
- GetDataFromDB(identifier: string): IEnumerator
+ OnMainMenuButtonClick(): void
+ OnBackButtonClick(): void

Figure 3.3.4.3: The UML notation of the “RetrieveHistory” script.

The explanations of methods:

1. Start()

   This method is called when the script is enabled. It initializes all the GameObjects and checks if all the necessary Firebase's dependencies are present and attempt to fix them if they are no present.
2. InitializeFirebase()

   This method connects the program to Firebase with URL.
3. StartListener()

   This method listens to the Firebase Realtime Database. Whenever the value is changed, it updates corresponding key-value pairs to the collection of "historyObject", and it deletes key-value pairs from the collection of "historyObject" if data in the database are removed.
4. Update()

   In Unity, this method is called once per frame to update GameObjects. In the history scene, the content to be showed in "HistoryPanel" are created within the algorithms of this method.
5. TaskWithParameters(string identifier)

   This method is triggered when the user clicks the "DetailButton". It retrieves the details of a particular row stated by 'identifier' from the database by calling GetDataFromDB in the coroutine. This method is paused for execution until the data is finished loading. After that, it hides "OBJ_HistoryPanel" and show "OBJ_DetailsPanel",
6. GetDataFromDB(string identifier)

   This method is called by above method "TaskWithParameters" by using the coroutine. It loads data from database and assigned the value to "Text_DetailsText". Once it finished loading the data, it restarts the "TaskWithParameters".
7. OnMainMenuButtonClick()

   This method is triggered when the user clicks the "ExitButton", which brings the user back to the main menu scene.
8. OnBackButtonClick()

   This method is triggered when the user clicks on the "BackButton" in "DetailsPanel". It shows "OBJ_HistoryPanel" and hides "OBJ_DetailsPanel".

3.3.4.4 The "DatabaseHandler" Script

This script is attached to the main game scene. As its name states, the main purpose of scripts is to work with databases. Once there is data to be recorded, it will upload the data to the Firebase Realtime Database, including the input values and the output values. In addition to this, this script also stores the input and output values as text files in the same folder as the program.

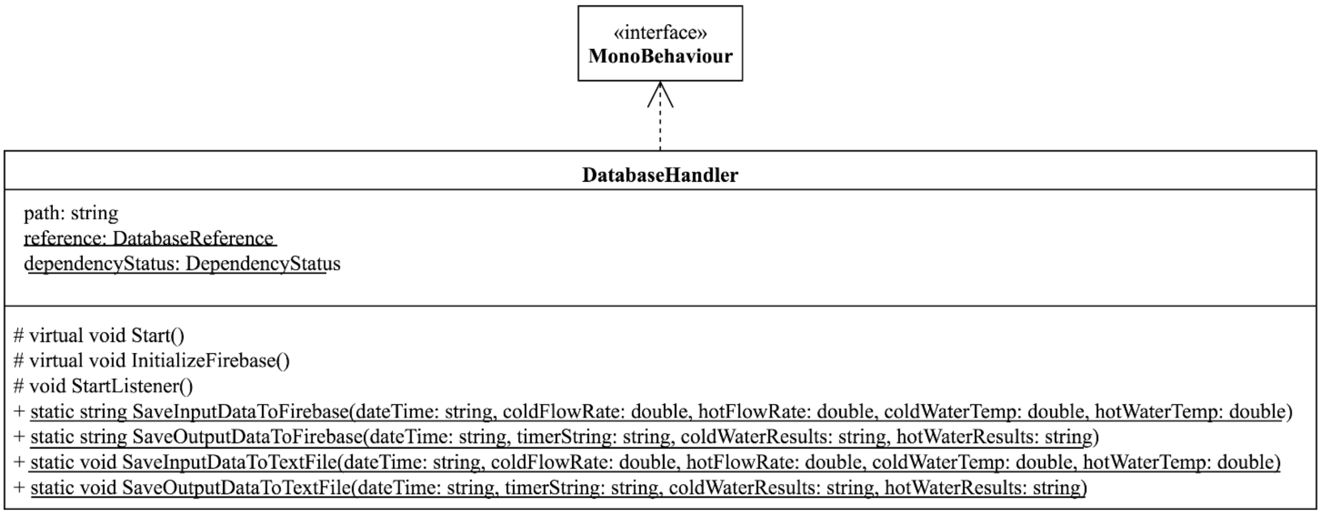

«interface»
MonoBehaviour
DatabaseHandler
path: string
reference: DatabaseReference
dependencyStatus: DependencyStatus
# virtual void Start()
# virtual void InitializeFirebase()
# void StartListener()
+ static string SaveInputDataToFirebase(dateTime: string, coldFlowRate: double, hotFlowRate: double, coldWaterTemp: double, hotWaterTemp: double)
+ static string SaveOutputDataToFirebase(dateTime: string, timerString: string, coldWaterResults: string, hotWaterResults: string)
+ static void SaveInputDataToTextFile(dateTime: string, coldFlowRate: double, hotFlowRate: double, coldWaterTemp: double, hotWaterTemp: double)
+ static void SaveOutputDataToTextFile(dateTime: string, timerString: string, coldWaterResults: string, hotWaterResults: string)

Figure 3.3.4.4: The UML notation of the "DatabaseHandler" script.

Explanation of the methods:

1. Start()
   This method is called when the script is enabled. It initializes all the GameObjects and checks if all the necessary Firebase's dependencies are present and attempt to fix them if they are no present.
2. InitializeFirebase()
   This method connects the program to Firebase with URL.
3. StartListener()
   This method sets the "reference" refers to the root of the database.
4. SaveInputDataToFirebase(string dateTime, double coldFlowRate, double hotFlowRate, double coldWaterTemp, double hotWaterTemp)
   This is a static method, it allows another script in the *Main Game Screen* to save input data to Firebase without having to create an instance of this class, such as passing parameters to "DatabaseHandler.SaveInputDataToFirebase()" will do the job.

5. SaveOutputDataToFirebase(string dateTime, string timerString,string coldWaterResults, string hotWaterResults)

   This is a static method, it allows another script in the *Main Game Screen* to save out data to Firebase without having to create an instance of this class, such as passing parameters to “DatabaseHandler.SaveOutputDataToFirebase ()” will do the job.
6. SaveInputDataToTextFile(string dateTime, double coldFlowRate, double hotFlowRate, double coldWaterTemp, double hotWaterTemp)

   This is a static method, it allows another script in main game screen to save input data to local storage as a text file without having to create an instance of this class, such as passing parameters to “DatabaseHandler.SaveInputDataToTextFile()” will do the job.
7. SaveOutputDataToTextFile(string timerString, string coldWaterResults, string hotWaterResults)

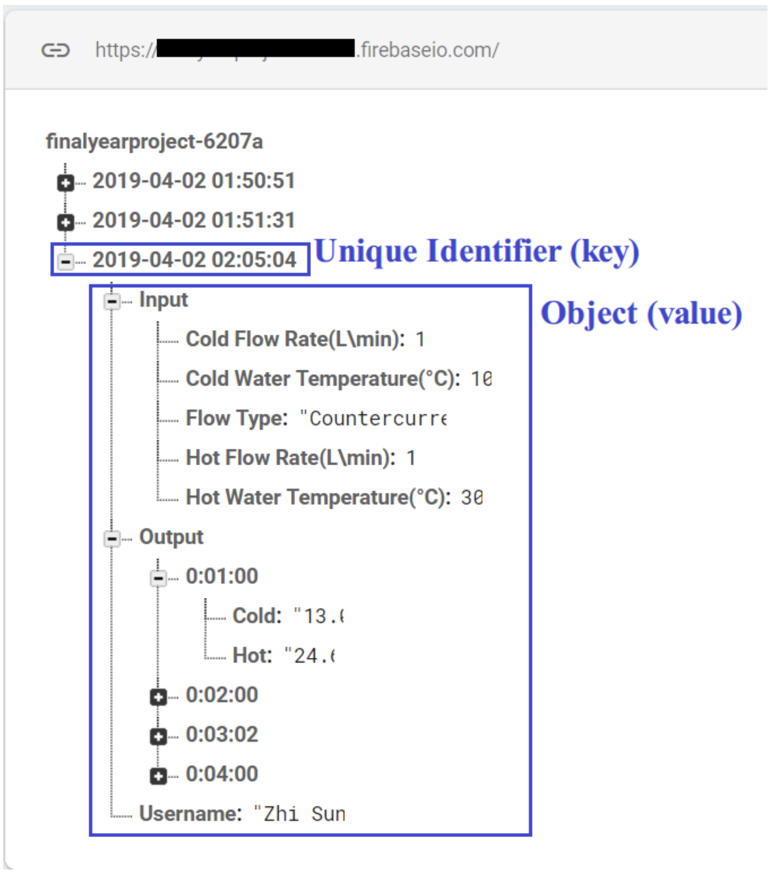
   This is a static method, it allows another script in main game screen to save input data to local storage as a text file without having to create an instance of this class, such as passing parameters to “DatabaseHandler.SaveOutputDataToTextFile()” will do the job.

3.3.5 The Structure of Data in the Firebase

All data in Firebase Realtime Database is stored as a large JSON object, which can hold key-value pairs. The value can be a string, number, array, Boolean, null or another object as long as it is associated with a unique key.

In the case of this project, since the date and time when the simulation starts change over time, in other words, the combination of data and time is different each time the simulation starts, so they can be selected as unique keys.

The value associated with each unique key was another object with three child objects: a *username key-value pair*, an *input object*, and an *output object*. In the *input object*, it contained 4 key-value pairs, and the *output object* contained a list of child objects.



```
https://[redacted].firebaseio.com/

finalyearproject-6207a
 ⊞ 2019-04-02 01:50:51
 ⊞ 2019-04-02 01:51:31
 ⊟ 2019-04-02 02:05:04
    ⊟ Input
        Cold Flow Rate(L\min): 1
        Cold Water Temperature(°C): 10
        Flow Type: "Countercurre
        Hot Flow Rate(L\min): 1
        Hot Water Temperature(°C): 30
    ⊟ Output
       ⊟ 0:01:00
           Cold: "13.
           Hot: "24.
       ⊞ 0:02:00
       ⊞ 0:03:02
       ⊞ 0:04:00
    Username: "Zhi Sun
```

Figure 3.3.5: The Structure of Data in the Firebase.

3.3.6 System Flowchart

3.3.6.1 Start Menu Scene

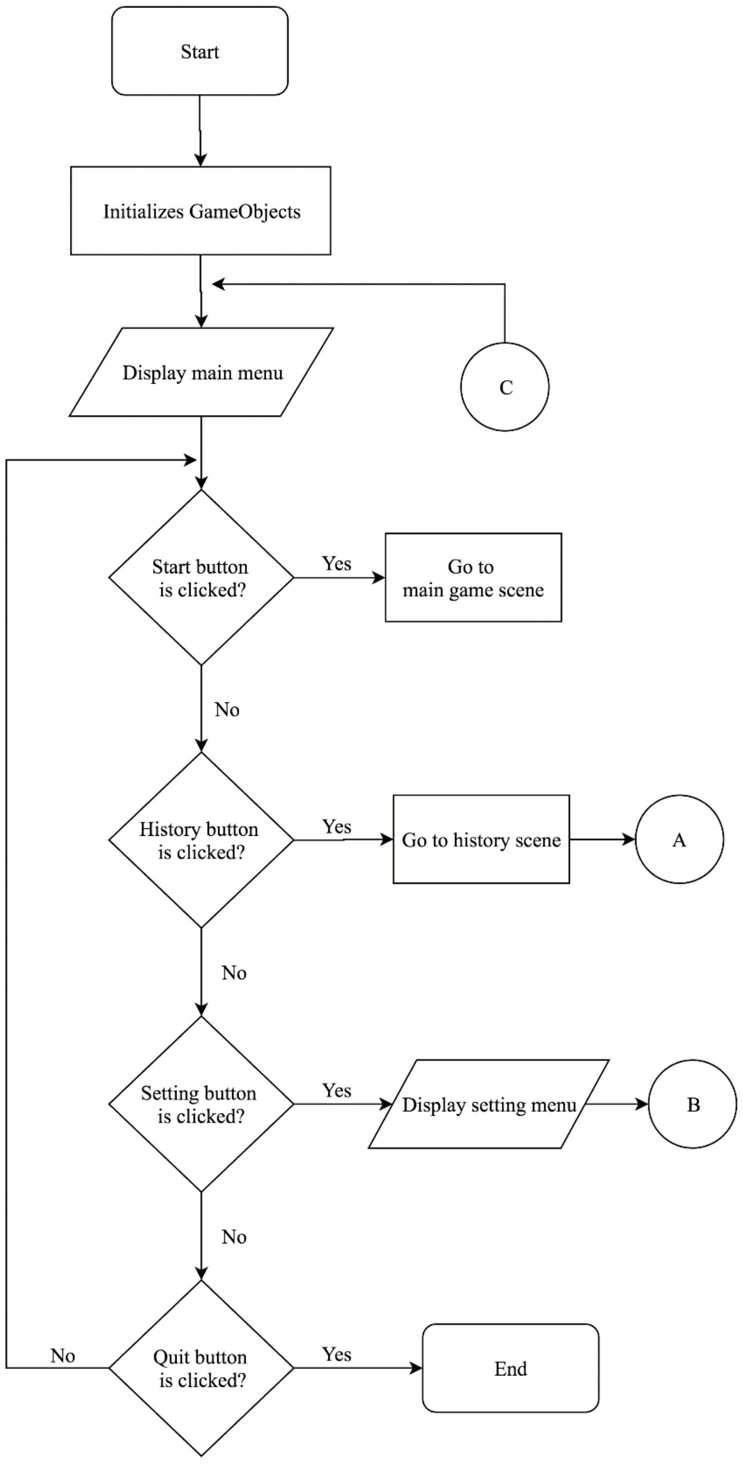


Figure 3.3.6.1: The flowchart of the Start Menu Scene.

3.3.6.2 History Scene

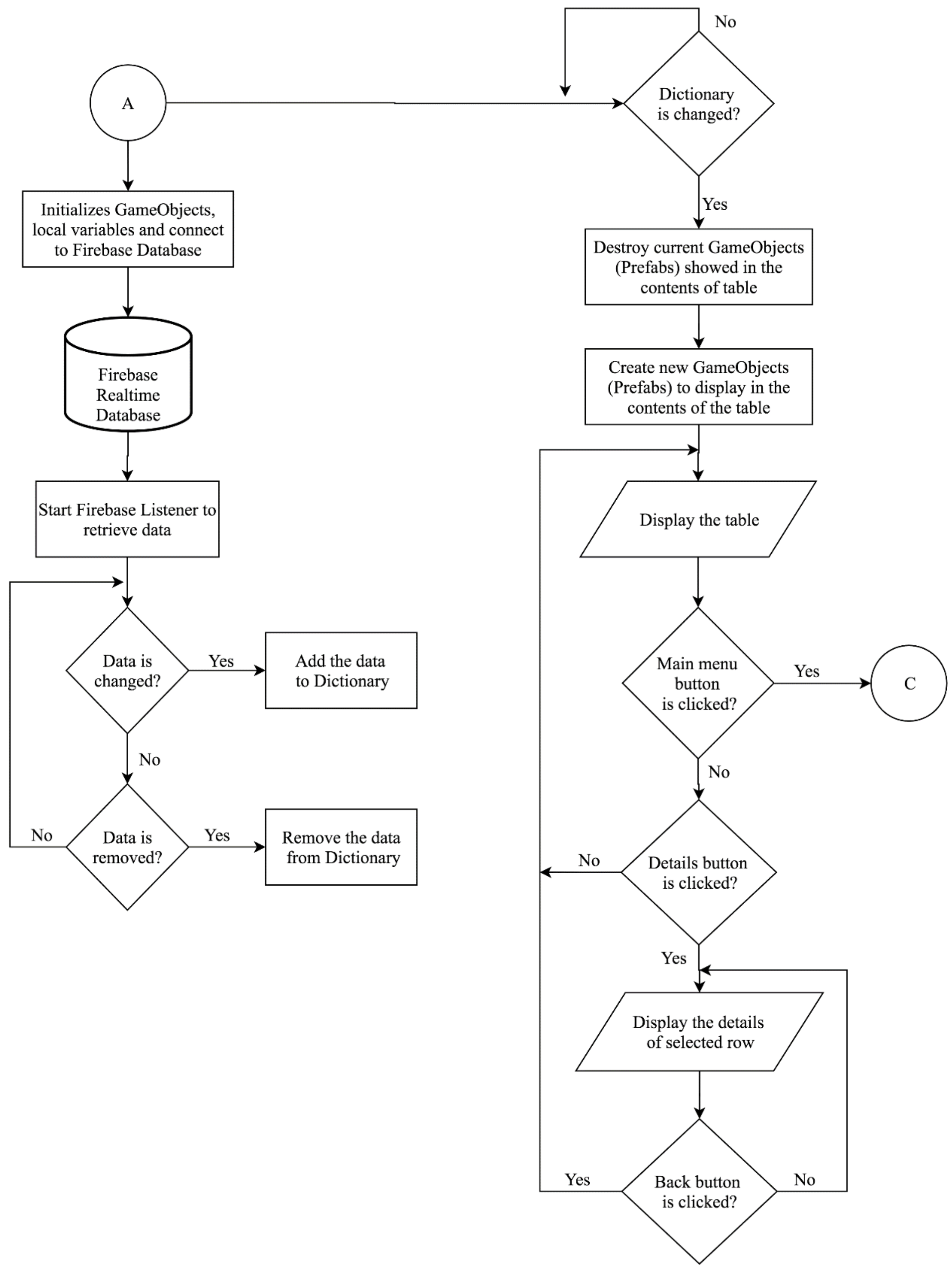

A
Initializes GameObjects, local variables and connect to Firebase Database
Firebase Realtime Database
Start Firebase Listener to retrieve data
Data is changed?
Yes
Add the data to Dictionary
No
Data is removed?
Yes
Remove the data from Dictionary
No
Dictionary is changed?
No
Yes
Destroy current GameObjects (Prefabs) showed in the contents of table
Create new GameObjects (Prefabs) to display in the contents of the table
Display the table
Main menu button is clicked?
Yes
C
No
Details button is clicked?
No
Yes
Display the details of selected row
Back button is clicked?
Yes
No

Figure 3.3.6.2: The flowchart of the History Scene.

3.3.6.3 Setting Menu

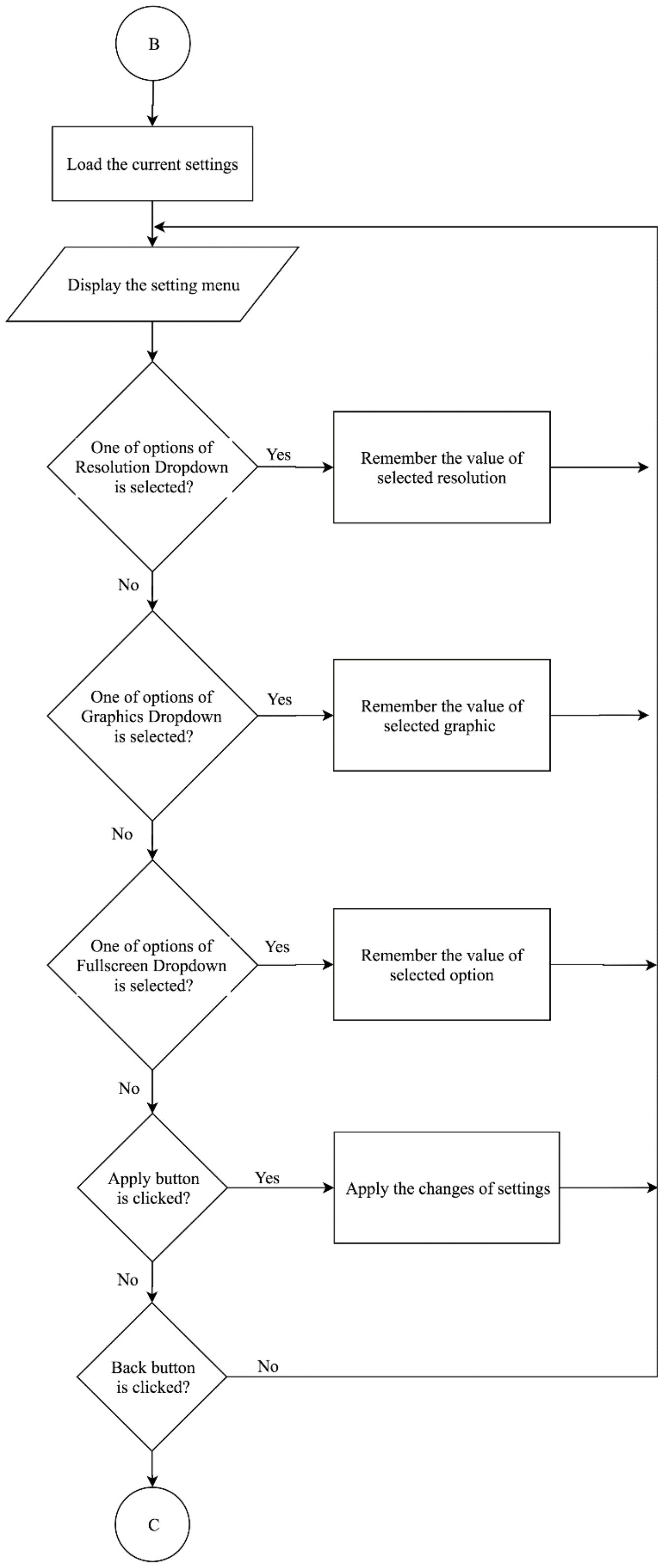

B
Load the current settings
Display the setting menu
One of options of Resolution Dropdown is selected?
Yes
Remember the value of selected resolution
No
One of options of Graphics Dropdown is selected?
Yes
Remember the value of selected graphic
No
One of options of Fullscreen Dropdown is selected?
Yes
Remember the value of selected option
No
Apply button is clicked?
Yes
Apply the changes of settings
No
Back button is clicked?
No
C

Figure 3.3.6.3: The flowchart of setting menu in the Start Menu Scene.

CHAPTER 4 VERIFICATION

In order to generate an accurate and reliable simulation program, the simulation program must be verified and validated after the development of the simulation program. The methods, tools, requirements, and plans for validating simulation program will be discussed in this chapter.

4.1 Methodology and Tools

Since the project was developed on the Windows 10 system platform, so the final product of the simulation program will be deployed on the Windows operating system. It must be built as an executable application in order to run on every device that has a Windows operating system installed. There is a list of items must be prepared for verification.

Items	Functionality
Windows OS computer	• To execute the final product of the simulation program.
File explorer	• To check the data stored at local storage after the simulation.
Unity editor	• To build the final product.
Internet browser	• To open Google Firebase Console.
Google Firebase Console	• To check the data uploaded to Firebase after the simulation.

Table 4.1: The items used for the verification of the final product.

4.2 Requirements

In order to ensure that the simulation program is rendered properly, there are certain requirements for the display device. First, the width and height of the screen are limited to the most common ratios 16:9. Second, the resolution of the display is suggested to:

Aspect Ratio	Resolutions	
16:9	• 1280×720 (Minimum) • 1600×900	• 1366×768 • 1920×1080 (Recommended)

Table 4.2: The requirements for the display device.

4.3 <u>Verification Plan</u>

The verification plan must be determined to test and verify the final product. It must go through all the steps that the user will go through while using the simulator. Therefore, the verification plan is divided into the following steps:

1. Check the function of the start menu

   This step has to check if the buttons are working. For example, the “Start” button will bring the user to proceed to start the simulation. Next, the “History” button can bring the user to the *History Scene*. Then, the “Setting” button can lead the user to configure the graphics settings. Last, the “Quit” button can terminate the simulation program. This step must check that all the buttons in the UI are working properly. For example, the “Start” button will guide the user to start the simulation. Next, the “History” button takes the user to the *History Scene* to view the data. The “Setting” button then guides the user through the configuration of the graphical settings. Finally, the “Exit” button can exit the simulation.
2. Check if the data can upload to cloud

   This step is done in the main game scene to check if all of the simulation data will be uploaded to the Firebase Realtime Database. The Google Firebase console will be opened by using any type of internet browser to see if the uploaded data is correct and well structured.
3. Check the function of history scene

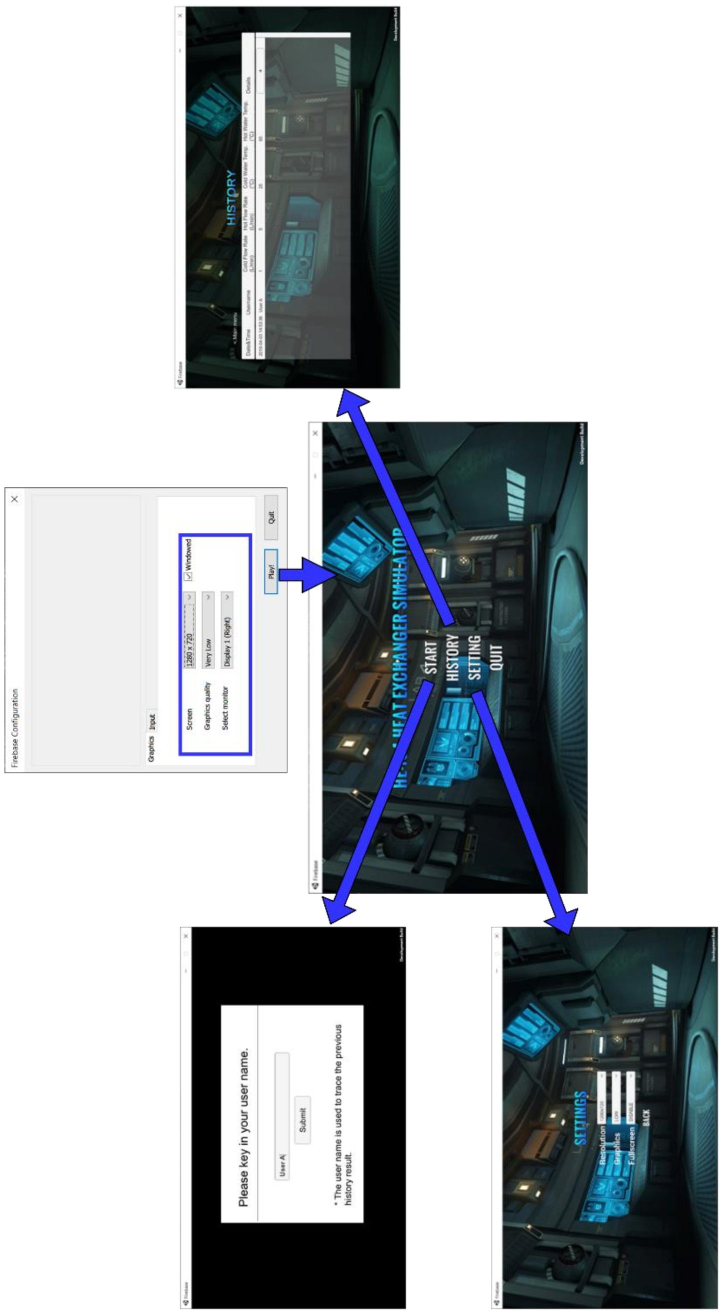
   In order for the simulation results to exist in the database and available, continue with this step after some simulations have been performed. This step will check the correctness of the retrieved data and check if the user can navigate through this scene.
4. Check the function of the local data storage

   This step is to check the simulation results stored in local storage after the simulation is done.

4.4 Implementation and Testing

1. Check the function of the start menu.

First, the user clicks the "Play" button to start the simulation and they will be taken to the start menu. When the user clicks “Start” or “History”, the screen changes to the corresponding view. Noted that when the simulation program is first started, the screen is set to windowed mode, the quality is set to low, and resolution is set to 1280 x 720. These settings will be applied as current settings. When the user clicks the "Settings" button, the current settings are loaded correctly and displayed in the settings menu.



Date&Time	Username	Cold Flow Rate (L/min)	Hot Flow Rate (L/min)	Cold Water Temp. (°C)	Hot Water Temp. (°C)	Details
2019-04-03 14:53:36	User A	1	8	25	95	+

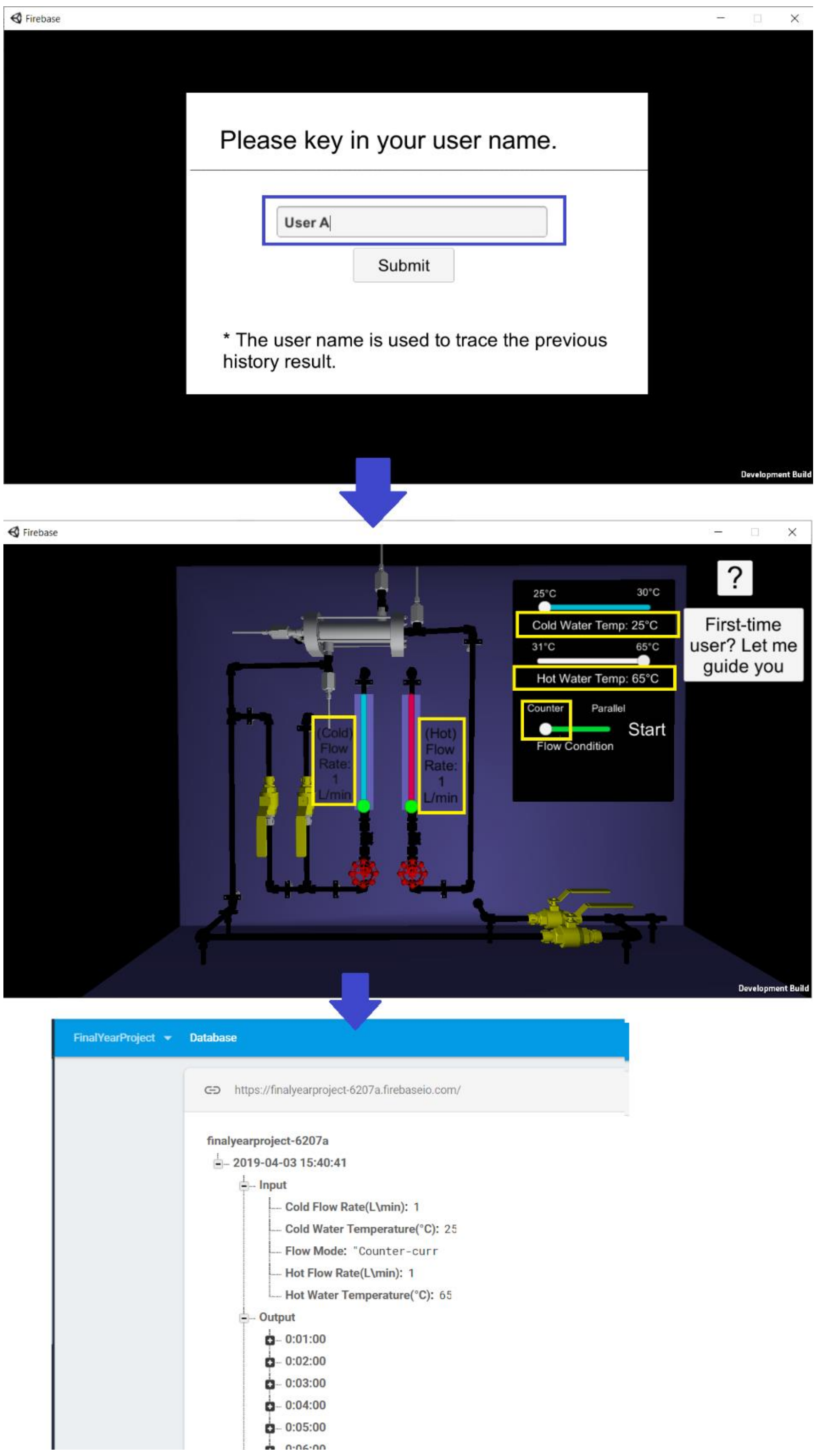
Screen: 1280 x 720 [✓ Windowed]
Graphics quality: Very Low
Select monitor: Display 1 (Right)

Please key in your user name.
User A
* The user name is used to trace the previous history result.

Figure 4.4.1: The start menu can work.

2. Check if the data can upload to the cloud.

“User A” is the username in this test. The cold-water inlet temperature is 25℃, the hot water inlet temperature is 65℃, and the cold water flow rate as well as the hot water flow rate is 1 L/min. After the user starts the simulation, these data are uploaded to the Firebase Database.


Firebase

Please key in your user name.

User A

Submit

* The user name is used to trace the previous history result.

Development Build

Firebase

25°C 30°C

Cold Water Temp: 25°C

31°C 65°C

Hot Water Temp: 65°C

Counter Parallel

Flow Condition

Start

?

First-time user? Let me guide you

(Cold) Flow Rate: 1 L/min

(Hot) Flow Rate: 1 L/min

Development Build

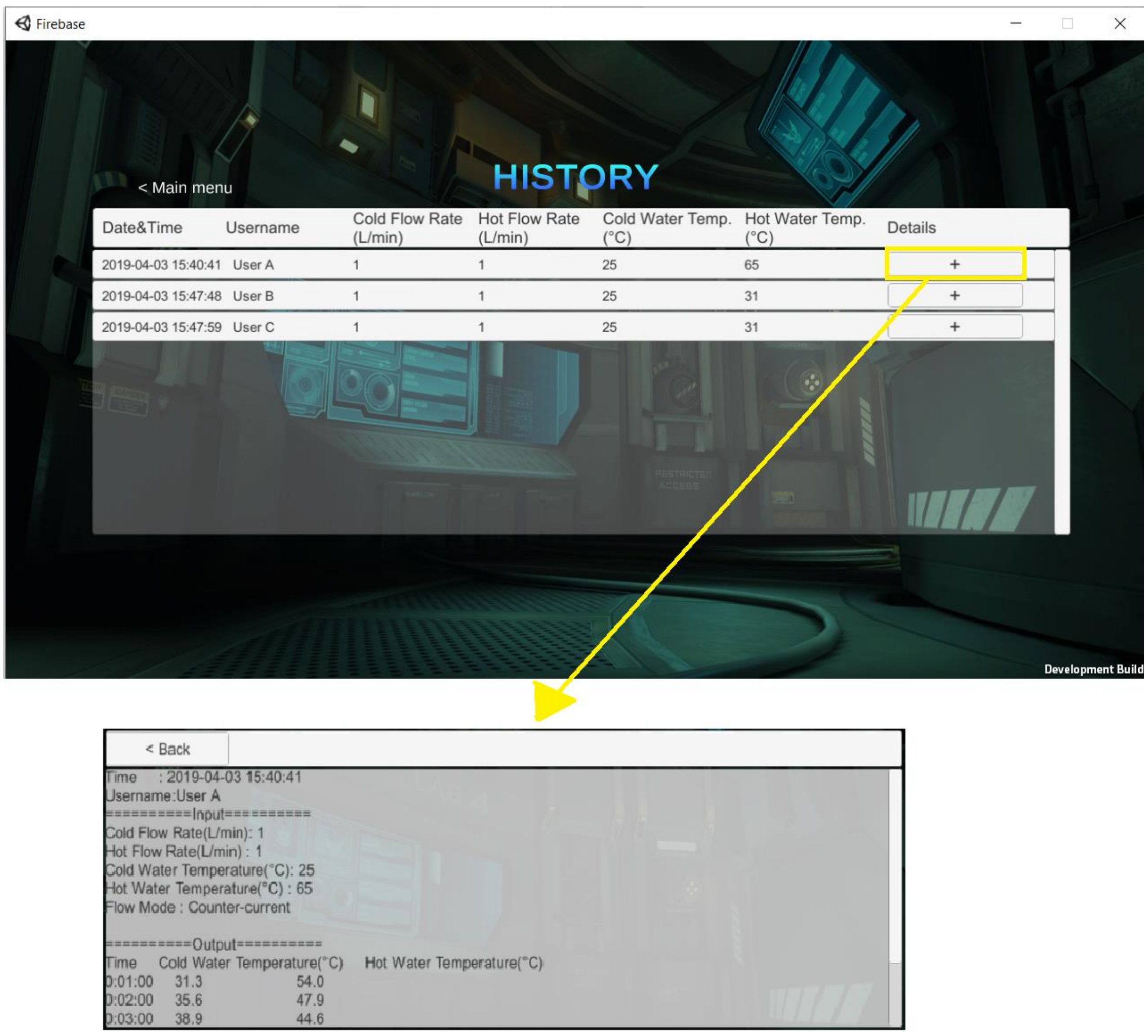
FinalYearProject Database

https://finalyearproject-6207a.firebaseio.com/

```
finalyearproject-6207a
 - 2019-04-03 15:40:41
    - Input
       Cold Flow Rate(L\min): 1
       Cold Water Temperature(°C): 25
       Flow Mode: "Counter-curr
       Hot Flow Rate(L\min): 1
       Hot Water Temperature(°C): 65
    - Output
       + 0:01:00
       + 0:02:00
       + 0:03:00
       + 0:04:00
       + 0:05:00
       + 0:06:00
```

Figure 4.4.2: The data can be uploaded and well structured.

3. Check the function of the *History Scene*.



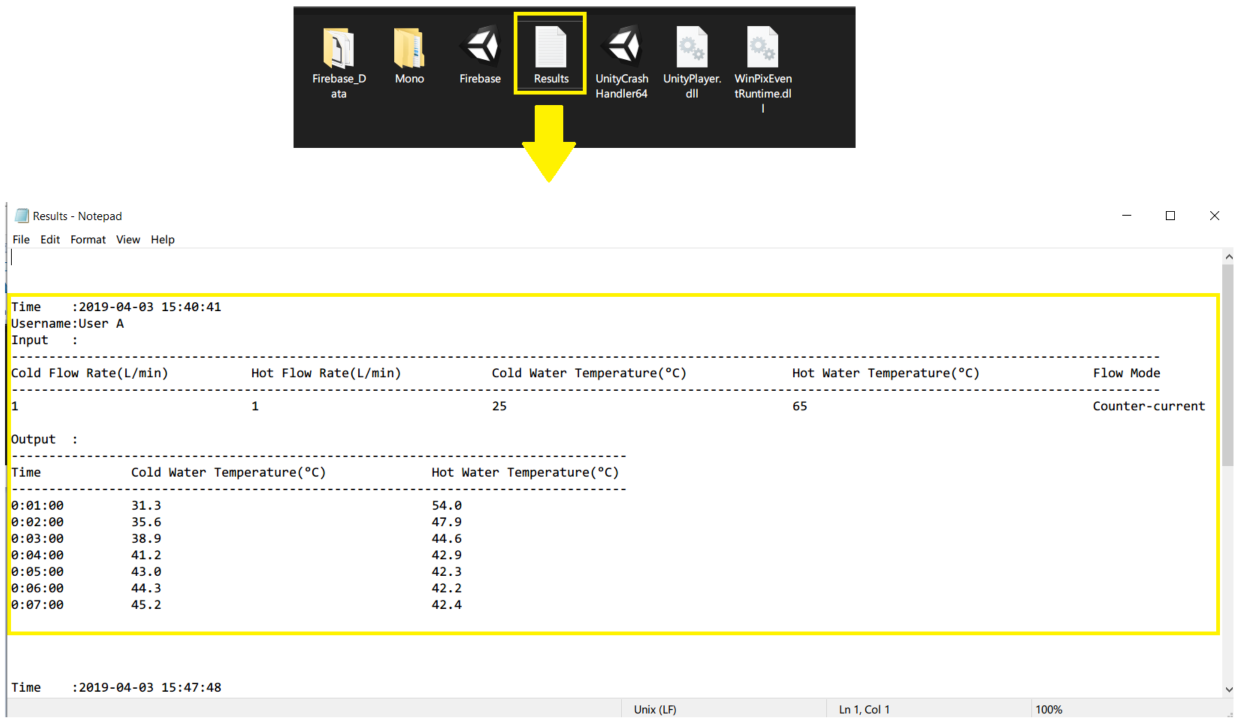
Date&Time	Username	Cold Flow Rate (L/min)	Hot Flow Rate (L/min)	Cold Water Temp. (°C)	Hot Water Temp. (°C)	Details
2019-04-03 15:40:41	User A	1	1	25	65	+
2019-04-03 15:47:48	User B	1	1	25	31	+
2019-04-03 15:47:59	User C	1	1	25	31	+

```
Time    : 2019-04-03 15:40:41
Username:User A
==========Input==========
Cold Flow Rate(L/min): 1
Hot Flow Rate(L/min) : 1
Cold Water Temperature(°C): 25
Hot Water Temperature(°C) : 65
Flow Mode : Counter-current

==========Output==========
Time    Cold Water Temperature(°C)   Hot Water Temperature(°C)
0:01:00   31.3                 54.0
0:02:00   35.6                 47.9
0:03:00   38.9                 44.6
```

Figure 4.4.3: The History Scene can work.

4. Check the function of the local data storage.



```
Time    :2019-04-03 15:40:41
Username:User A
Input   :
--------------------------------------------------------------------------------------------------------------------------------------------------------
Cold Flow Rate(L/min)          Hot Flow Rate(L/min)          Cold Water Temperature(°C)          Hot Water Temperature(°C)          Flow Mode
--------------------------------------------------------------------------------------------------------------------------------------------------------
1                              1                             25                                  65                                 Counter-current

Output  :
--------------------------------------------------------------------------------
Time            Cold Water Temperature(°C)          Hot Water Temperature(°C)
--------------------------------------------------------------------------------
0:01:00         31.3                                54.0
0:02:00         35.6                                47.9
0:03:00         38.9                                44.6
0:04:00         41.2                                42.9
0:05:00         43.0                                42.3
0:06:00         44.3                                42.2
0:07:00         45.2                                42.4


Time    :2019-04-03 15:47:48
```

Figure 4.4.4: The simulation can save as text file in local storage.

CHAPTER 5 IMPLEMENTATION ISSUES AND CHALLENGES

There are several implementation issues and challenges faced during the development of this simulation program. The first challenges are the exploration and exploitation of a new field. Before creating a simulator, it takes a lot of effort to traverse through unfamiliar areas. A certain understanding of heat exchangers has to be acquired first. Besides, it takes time to learn how to use Unity. The concept in Unity like ‘GameObject’, ‘Component’, script and, so on is totally new to learn.

Next, the challenge is to use Unity with Firebase. The first Firebase SDK merged with Unity was released on November 7, 2016. There are limited resources for the newbie to practice or refer to since it is a new technology in the market of Unity. Moreover, Firebase update rate is very high and therefore the developers need to catch up with the new things frequently.

Finally, the design of the user interface is another issue that matters to developers. While Unity does provide a lot of assets in the Unity Asset Store to help developers create their GameObjects, many of them require developers to pay for it, and as for free assets, they all look simple and even useless for this project. Therefore, simple built-in UI objects are used to create the user interface for this simulator program.

CHAPTER 6 CONCLUSION

The heat exchanger unit is a useful teaching device for engineering students to learn and study the basic principles of heat transfer. The problem of conducting heat exchanger experiments was discussed in chapter 1. Meanwhile, chapter 2 discussed the heat exchanger system in term of the type, the past modelling, and the mathematical model. In addition, an overview of Firebase was also introduced. Therefore, a heat exchanger simulator with a real-time database was developed to solve experiment problems in this project.

Among the many real-time engine development platform, Unity was selected as the development tool. The development work was started after task partitioning. A top-to-down level explanation of the system design was discussed in chapter 3. The merger of Firebase and Unity was an area to be studied and done in this project.

Through the simulator created in this project, the user (e.g. engineering students) can learn the basic principles of heat transfer without physical heat exchangers. It not only eliminates the cost and danger of physical experiments but also improves the efficiency of recording experiment data. In addition, the user of this simulator can take advantage of obtaining the past simulation data anytime and anywhere as long as they have this simulator.

BIBLIOGRAPHY

Alsalemi, A. et al., 2017. Real-Time Communication Network Using Firebase Cloud IoT Platform for ECMO Simulation. *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData),* pp. 178-182.

Amirkhani, S. & Nahvi, A., 2016. Design and implementation of an interactive virtual control laboratory using haptic interface for undergraduate engineering students.

Bendapudi, S., Braun, J. E. & Groll, E. A., 2004. Dynamic Modeling of Shell-and-Tube Heat Exchangers: Moving Boundary vs. Finite Volume.

Berbakov, L. et al., 2017. Android application for collaborative mapping in emergency situations. *2017 25th Telecommunication Forum (TELFOR),* pp. 1-4.

CORREA, D. J. & MARCHETTI, J. L., 1987. Dynamic Simulation of Shell-and-Tube Heat Exchangers. *Heat Transfer Engineering,* pp. 50-59.

Firebase, n.d. *Firebase Realtime Database | Store and sync data in real time.* [Online] Available at: https://firebase.google.com/products/realtime-database/ [Accessed August 2018].

Fritzson, P., 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* s.l.:John Wiley & Sons.

GADDIS, E. S. & SCHLüNDER, E. U., 1979. Temperature Distribution and Heat Exchange in Multipass Shell and Tube Exchanger with Baffles. *Heat Transfer Engineering,* Volume 1, pp. 42-52.

Hellborg, J., 2017. Modelling of shell and tube heat exchangers.

Kerpelman, T., 2016. *Announcing Firebase for Unity.* [Online] Available at: https://firebase.googleblog.com/2016/11/announcing-firebase-for-unity.html

LUNSFORD, K. M., 2016. Increasing Heat Exchanger Performance. p. 13.

M.R.Ansari & V.Mortazavi, 2006. Simulation of dynamical response of a countercurrent heat exchanger to inlet temperature or mass flow rate change. *Applied Thermal Engineering,* 26(17-18), pp. 2401-2408.

Martikka, H., 2004. COMPARISON STUDY OF TWO COMPETING MODELS OF AN ALL MECHANICAL POWER TRANSMISSION SYSTEM.

Martin-Villalba, C., Urquia, A. & Dormido, S., 2009. Visualization and interactive simulation of Modelica models for control.

Shah, R. K. & Sekulić, D. P., 2007. *Classification of Heat Exchangers.* s.l.:s.n.

Sunil, A., S., P. & K.B, G., 2014. DESIGN OF SHELL AND TUBE HEAT EXCHANGER USING COMPUTATIONAL FLUID DYNAMICS TOOLS.

Tamplin, J., 2014. *Firebase is Joining Google!.* [Online] Available at: https://firebase.googleblog.com/2014/10/firebase-is-joining-google.html

Woodford, C., 2018. *Heat exchangers.* [Online] Available at: https://www.explainthatstuff.com/how-heat-exchangers-work.html [Accessed 2018].

APPENDIX

An Educational Simulator with Realtime Database

INTRODUCTION

A heat exchanger simulator with real-time database will be developed. With the support of the database, the recording and analysis of test data will be better improved.
Students can simply enter some values into the simulator and then get a transient response to the temperature.

METHOD

POWERED BY unity

Built with Firebase

DISCUSSION

It is a customized simulator

→Unity allows developers to create their own objects. In other words, developers can use it to create a heat exchanger object!

It is support by Firebase

→Firebase now supports Unity! The simulator can take advantage of Firebase to bring more functionality!

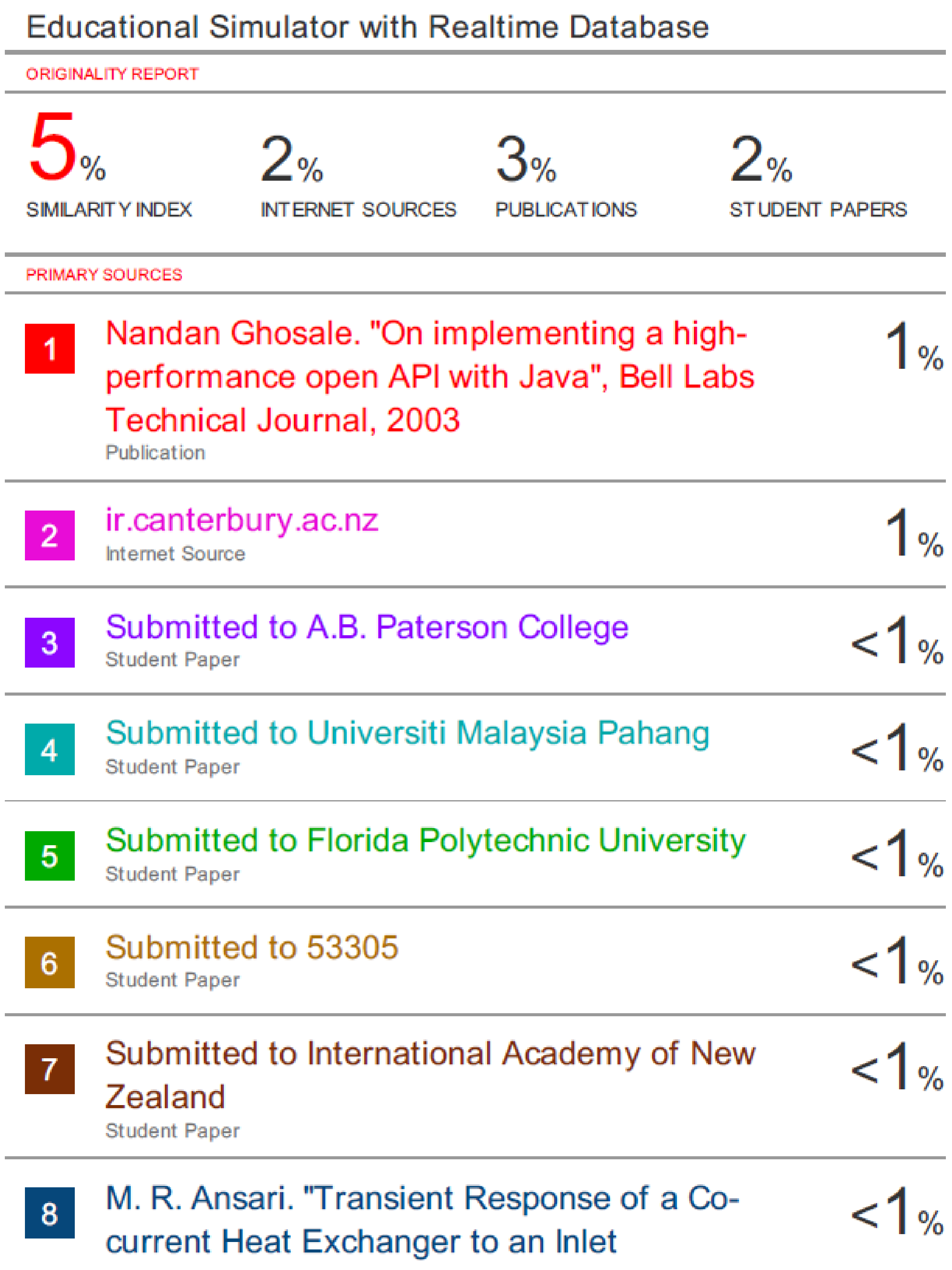
CONCLUSION

The simulator is designed to enhance the learning curve of engineering students. Students can use this simulator to gain knowledge of the heat transfer principle.

PEOPLE

Project Supervisor:
Dr. Chang Jing Jing
Project Developer:
Tok Zhi Sung

Turnitin Checked Result



Educational Simulator with Realtime Database

ORIGINALITY REPORT

5%	2%	3%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Nandan Ghosale. "On implementing a high-performance open API with Java", Bell Labs Technical Journal, 2003 Publication	1%
2	ir.canterbury.ac.nz Internet Source	1%
3	Submitted to A.B. Paterson College Student Paper	<1%
4	Submitted to Universiti Malaysia Pahang Student Paper	<1%
5	Submitted to Florida Polytechnic University Student Paper	<1%
6	Submitted to 53305 Student Paper	<1%
7	Submitted to International Academy of New Zealand Student Paper	<1%
8	M. R. Ansari. "Transient Response of a Co-current Heat Exchanger to an Inlet	<1%

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	
ID Number(s)	
Programme / Course	
Title of Final Year Project	

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index:_________ % **Similarity by source** Internet Sources: _______________% Publications: ___________ % Student Papers: __________ %	
Number of individual sources listed of more than 3% similarity: _________	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

______________________________	______________________________
Signature of Supervisor	Signature of Co-Supervisor
Name: _________________________	Name: _________________________
Date: __________________________	Date: __________________________

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **<u>Project II</u>**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 2
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - Planned the refinements that need to be done from the prototype in FYP 1.
2. WORK TO BE DONE - Discuss with Kong Yee Kian how to cope with each other.
3. PROBLEMS ENCOUNTERED - So far so good
4. SELF EVALUATION OF THE PROGRESS - Nothing

Supervisor’s signature

Student’s signature

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **<u>Project II</u>**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 4
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - Task partitioning was done after discussed with Kong Yee Kian.
2. WORK TO BE DONE - Create the user interface and write the basic script.
3. PROBLEMS ENCOUNTERED - So far so good.
4. SELF EVALUATION OF THE PROGRESS - Nothing.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **Project II**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 6
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - Start menu and setting menu were done.
2. WORK TO BE DONE - Try to link my scene with Kong Yee Kian's scene. - Refine the task partitioning.
3. PROBLEMS ENCOUNTERED - Full-screen mode cannot work because the library has been updated.
4. SELF EVALUATION OF THE PROGRESS - Need to speed up the progress.

_______________________________ _______________________________

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **Project II**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 8
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - The merger of scenes was done. - The data generated in the main game scene can be uploaded to Firebase.
2. WORK TO BE DONE - Retrieve the data from Firebase and display in history scene.
3. PROBLEMS ENCOUNTERED - The firewall of the router blocks the outgoing data from Unity. - The version of Unity installed in my PC and Kong Yee Kian's pc is different, causing some compatible problems.
4. SELF EVALUATION OF THE PROGRESS - Again, need to speed up the progress.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **<u>Project II</u>**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 10
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - The history scene was done.
2. WORK TO BE DONE - Merge my project with Kong Yee Kian's project. - Try to build the final product.
3. PROBLEMS ENCOUNTERED - The resolution of the monitor affects the rendering of GameObjects, causing GameObjects malformation and misplacement. - The "TextMess Pro" asset caused a lot of errors when combined with each other's projects.
4. SELF EVALUATION OF THE PROGRESS - The project is behind schedule. Need to speed up.

_____________________________ _____________________________

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

*(Project I / **Project II**)*

Trimester, Year: Year 3 trimester 3	**Study week no.:** 11
Student Name & ID: Tok Zhi Sung - 1605148	
Supervisor: Dr. Chang Jing Jing	
Project Title: Educational Simulator with Realtime Database	

1. WORK DONE - The overall of the project was done.
2. WORK TO BE DONE - Finalize the simulation program and write a report.
3. PROBLEMS ENCOUNTERED - So far so good.
4. SELF EVALUATION OF THE PROGRESS - Nothing.

Supervisor's signature

Student's signature

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	
Student Name	
Supervisor Name	

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Cover
	Signed Report Status Declaration Form
	Title Page
	Signed form of the Declaration of Originality
	Acknowledgement
	Abstract
	Table of Contents
	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
	List of Abbreviations (if applicable)
	Chapters / Content
	Bibliography (or References)
	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
	Poster
	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. ________________________ (Signature of Student) Date:	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. ________________________ (Signature of Supervisor) Date: