The final report must begin with the project information specified in Annex I. After that, a project report must follow, whose structure should be as specified next. A length of one page per section is recommended, except for section 3, for which recommendation is two pages.

Report Contents

1. INTRODUCTION

General description of the project and brief summary of the report.

In this project, I developed a Middleware that is used for recognizing a person's skeleton using different 3D motion capture cameras (either the Kinect for Xbox One or Kinect Azure, both created by Microsoft) and can send this information to a Unity exercise game in real-time.

Last semester, my objective was to create a middleware that would be able to communicate with Azure Kinect using its SDK (software development kit) and send data to a Unity game using existing UDP communication shame. Additionally, I was also able to add backward compatibility with Kinect One. For technical reasons, I was also forced to write the entire user interface from scratch.

During this semester I changed the data transfer format, from a custom system which was very data efficient to JSON which allows us to easily decode and encode data which makes it easier to add additional information that can be sent in the future. This change was made because we wanted to add the ability to send information about the heart rate that was to be obtained from a programmable smartwatch. This work was thought to be done by another internship student but could not have been finished during this semester due to technical problems.

Moreover, I added a feature for the user's login, which allows the connection to the database, to obtain the user's configuration parameters and upload the results of the gaming exercises. The middleware saves these results in the memory of the device and when the user finishes the game and closes the software, all user's data are sent to the server where they are stored.

Additionally, I added a skeleton overlay feature within the camera tab, this is allowing the users to visualize the currently detected skeletons. This provides users with real-time insight into what exactly is detected by the device. This feature is supported both by the Kinect One and Azure.

I also devoted a lot of attention to the software architecture of the entire application, improving structure and coding efficiency. The program has been divided into many smaller classes that have very precise tasks. This is very important because it improves the readability of the code and allows for easier development and maintenance.

The application has also been localized, which means that it is available in both English and Spanish. The system is quite flexible so if there would be a necessity to add more languages in the future it will be very easy.

2. CONTEXT

UPM centre in which the Project has been carried out, main responsibilities, interaction with other people (students, researchers, etc.), lab description (if appropriate), etc.

The project described in this report was carried out at the UPM centre CITSEM (Software and Multimedia Systems Technologies for Sustainability Research Centre). This centre aims to strengthen and promote R&D&I and attract research talent at the UPM South Campus.

As a student working on this project, my main responsibilities included creating a software architecture that will be simple in further development, maintenance, and adding new functionalities. I was also responsible for developing this software and debugging it. Moreover, I had to keep track of my progress and report regularly to my supervisor with whom I had weekly meetings to discuss my progress and future goals.

The laboratory in which the project was carried out was well-equipped with a variety of tools and resources for software development, including computers, software licenses, and other necessary equipment like Xbox One Kinect, and Azure. The laboratory also had a supportive and collaborative atmosphere, which was helpful for working on the project.

In addition, this semester I proposed that we start using Git for version control using the GitHub platform. It was supposed to make it much easier for me to work together with another student who was also supposed to develop the middleware with new functionalities. However, due to technical problems, this did not happen, but the project is on the Microsoft platform, which will facilitate future group work on this project.

Overall, the UPM Centre provided an ideal environment for carrying out this project, with excellent resources and support for students and researchers.

3. PROJECT OBJECTIVES AND TASKS

Completed tasks, with indication on how they have contributed to the fulfilment of the project objectives, problems and difficulties, proposed solutions, etc.

The main goal I set myself for this semester was to re-organize the code of the middleware and the belonging functions of the Unity Asset that receives the skeleton data, dividing them into smaller parts with precisely defined tasks. I focused on this because it is clear to me that this software needs further development and the biggest barrier I had to face when I started working on this project was understanding the dependencies that are in it. So, I would like to make someone future work easier on this project.

3.1. Changes and enhancements performed in the middleware:

The original project consisted of a main file of around 1000 lines of code and several others that were of a similar length. As a result, changing the smallest thing in the code by a person like me who does not know exactly how everything works together caused a lot of errors.

When I was working on the project architecture, I tried not to exceed 200 lines in any of the classes, and rather keep it within 100-150 so that each class is easy to understand and maintained.

Moreover, the original project contained a lot of logic in the screens. Personally, I think it's not the best practice, so I tried to remove as much logic as possible from the screens. (.xaml.cs files) and move it to separate classes. In the end, the screens should only contain the logic responsible for displaying information and calling external functions as presented in Figure 1. My actions result from the fact that if the user interface needs to be changed in the future for some reason, the code can still be used because it does not depend so much on how the interface works.

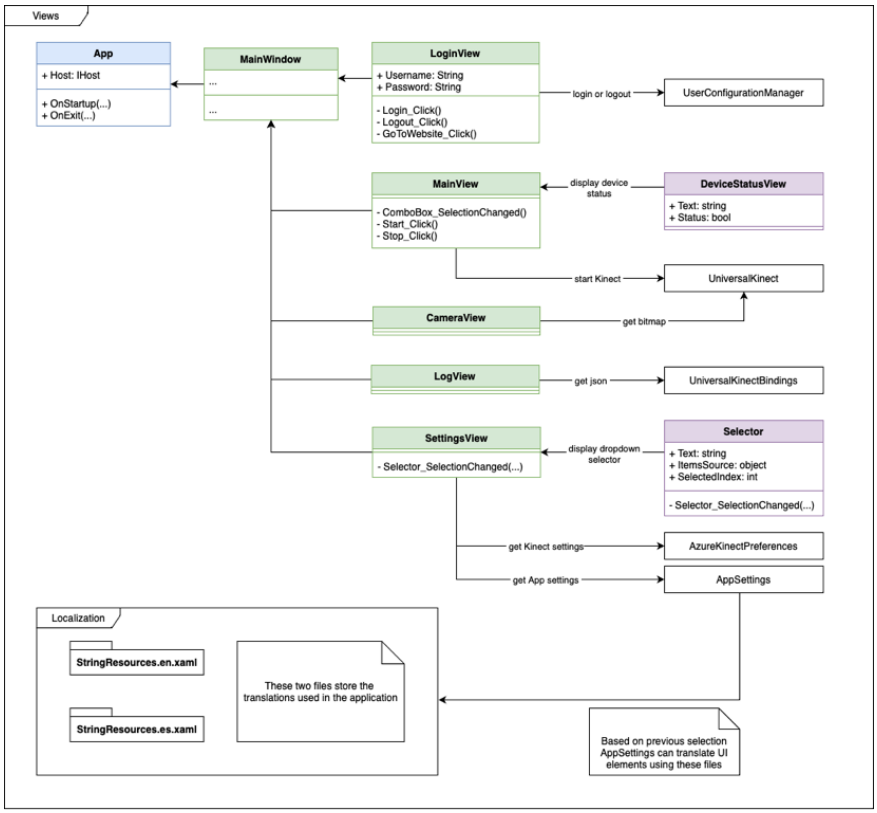

Figure 1 UI Architecture

So now we have ***App*** that initializes all the necessary class. Among others ***MainWindow***, that manages the logic for displaying tabs. The application has 5 tabs:

- ***LoginView*** - for login and opening administrator webpage,
- ***MainView*** - for choosing Kinect device and monitor its status,
- ***CameraView*** - for displaying camera image and skeleton overlay,
- ***LogView*** - for showing JSON that is transmitted,
- ***SettingsView*** - for setting App and Kinect Azure settings.

For this there are 2 views ***DeviceStatusView*** and ***Selector***. The first shows the device status and is used in the main screen. The second is to create a dropdown selector with label and is used in settings tab.

Additionally, we have 2 ***StringResources*** files for storing app localization which are used in all the views.

I also created separated classes which only task is to store constants used in our application. This allows us to easily change these variables in one place, which makes it easier for us to maintain the code in the future. Currently there are 3 classes:

- ***Paths*** - for storing local wipes that we use to save and read files,
- ***URLs*** - for storing URLs for webpages and for API requests,
- ***UDPPorts*** - for setting ports used by UDP protocol.

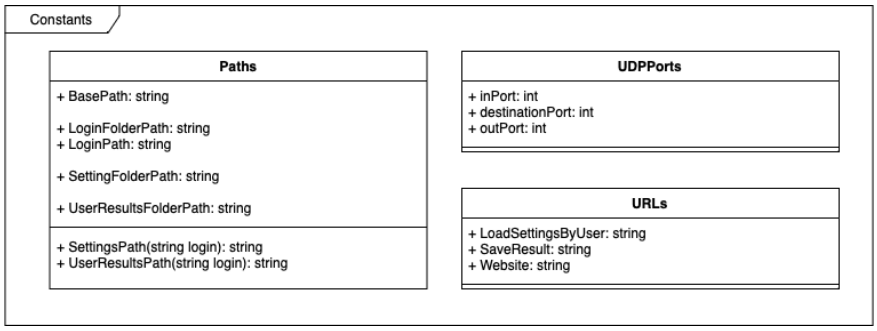

Figure 2 Classes storing constant in the application

Moreover, I added settings for the Azure Kinect (***AzureKinectPreferences*** class), so the user can define the orientation, processing mode, resolution, and number of frames per second that are displayed in UI (user interface). After this change and a few small improvements, the current architecture of the middleware looks like on outlined in Figure 3.

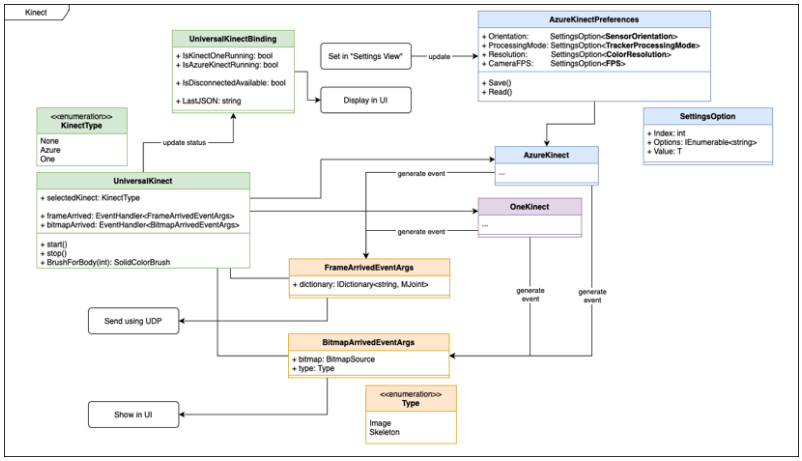

Figure 3 Kinect Architecture

Due to the implementation of data transfer from the server to the Unity game, I had to add additional data models which are shown in the diagram in Figure 4. There is a ***MJoint*** is a class for storing a joint representation it is using Euler class for converting Euler angle representation to Quaternion. Two classes for representing web response: ***WebLoginResponse*** and ***WebSingleResultResponse*** both have ***Status*** Enum for representing possible responses. ***LoadSettingResponse***, ***Game***, ***Exercise*** and ***Setting*** this is data which middleware receives from the server. ***UserResults*** and ***ExerciseResult*** are representation of user exercises results and are sent to the server.

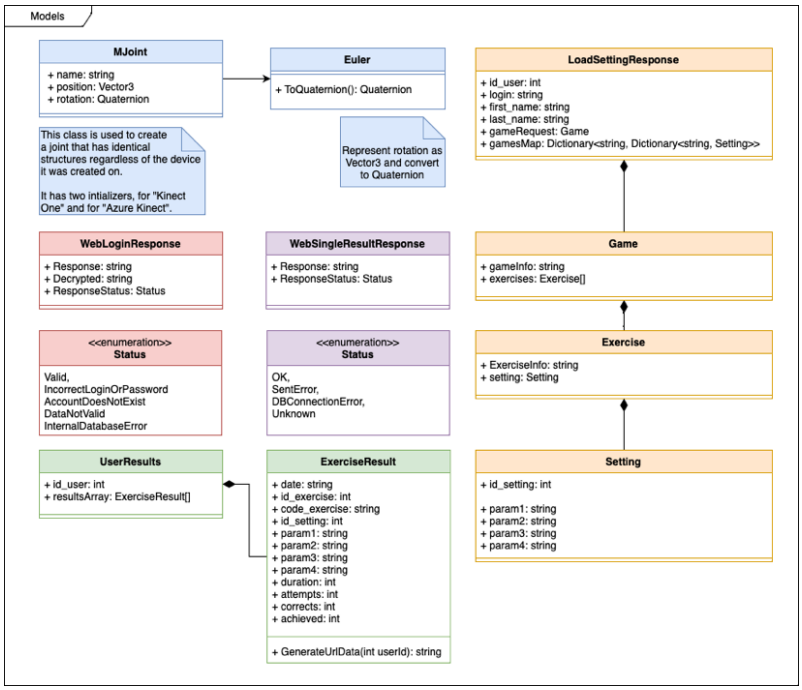

Figure 4 Data models

All this data is transmitted to the Unity game using the UDP connection, so it also needed some changes that are presented in Figure 5.

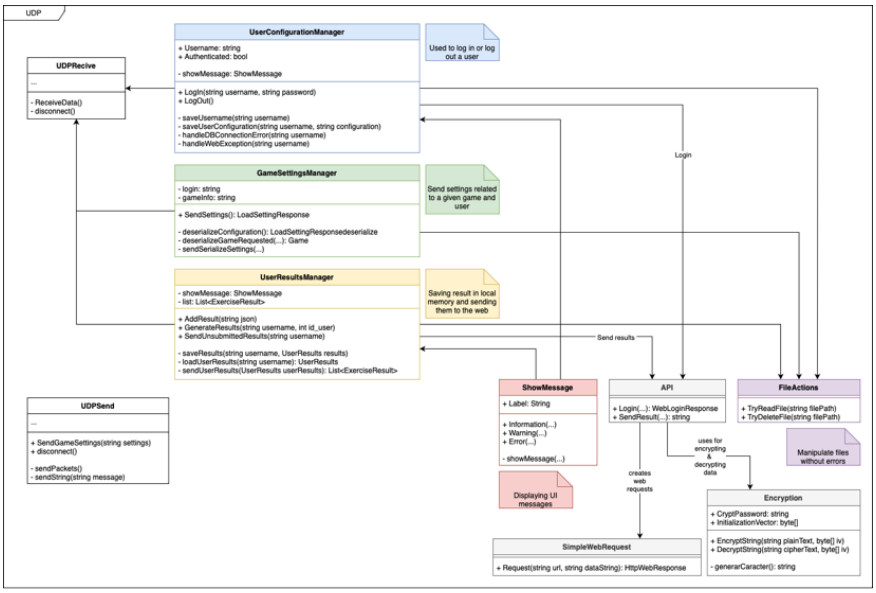

Figure 5 UDP

There are 2 main classes: ***UDPSend*** which does not need any additional dependencies and ***UDPRecives*** which uses 3 additional dependencies:

- ***UserConfigurationManager*** *– used for login and logout the user.* ***UDPRecives*** *use it to check if the user is sign in and what is his/her username,*
- ***GameSettingsManager*** *– used for generating game settings that are send to Unity,*
- ***UserResultsManager*** *– used for storing and sending user results.*

Additionally, there are 3 helper classes:

- ***ShowMessage*** – for showing user interface box dialog for displaying crucial information,
- ***API*** – for making predefine API requests using 2 subclasses ***SimpleWebRequest*** and ***Encryption***,
- ***FileAction*** – for managing files without errors.

3.2. Changes and enhancements performed in the Unity Asset:

On the Unity side, most of the code was rewritten to make it more maintainable and readable, the transfer implementation was changed from a custom format to JSON. The current architecture looks like on Figure 6.

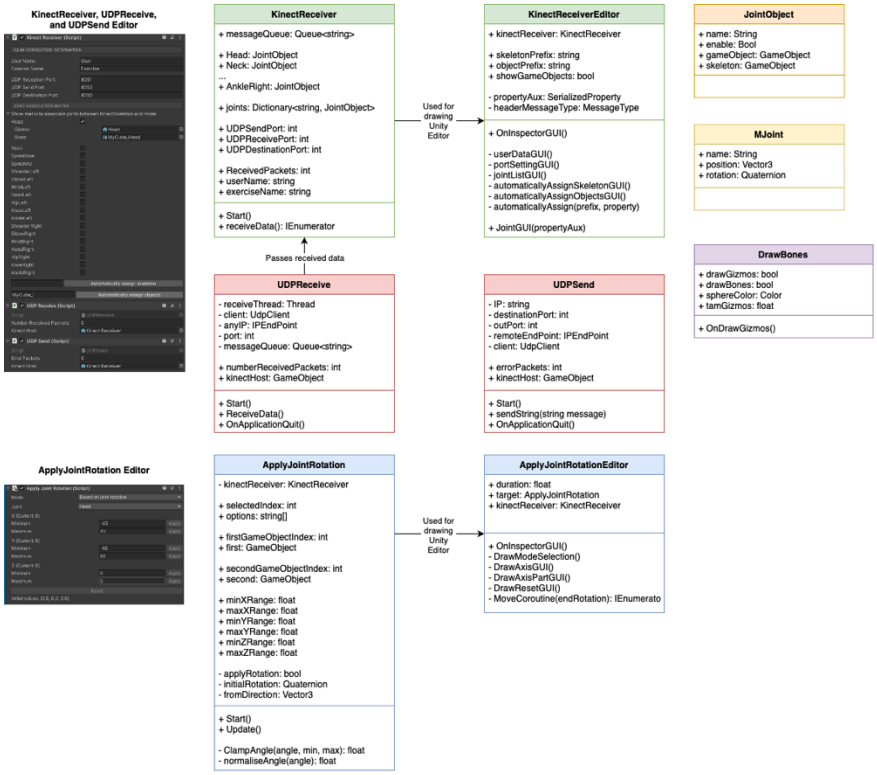

Figure 6 Unity architecture

In a few sentences, architecture works as follows: the data is obtained from the middleware using the ***UDPReceive*** class (which use UDP for communication). The data is sent to the ***KinectReceiver*** where it is converted from JSON to a dictionary of ***MJoint***. ***MJoint*** is an object which stores information about a position and rotation of a joint. Then, the obtained data is assigned to the objects that we set using the Unity Editor (***KinectReceiverEditor*** class menage the editor UI in Unity) which with the help of the ***DrawBone*** class we can present a visualisation of the skeleton. Information about physical objects in Unity are stored by ***JointObject***. In addition, we have the ***ApplyJointRotation*** class, that allows us to assign any joint rotation to any type of object in Unity, we will use it to assign rotation to character joints in the game.

4. TECHNOLOGIES AND EQUIPMENT

Description of algorithms, equipment, software, and other tools used during the project.

For this project, I used a variety of tools and technologies to develop this, these included:

- Visual Studio: This integrated development environment (IDE) was used for writing and debugging the code for our application and writing code in Unity.
- app.diagrams.net: This is an online application that I used for creating all the diagrams that are in this rapport.
- Fork: for managing Git and committing new changes on GitHub.
- Wireshark: Due to the low readability of the original project, I used this packet analyser, which allowed me to intercept the UDP request and to see what exactly the exchange between middleware and Unity looks like, this allowed me to easily reconstruct this system in my application.
- Unity: This software was used for creating an asset that receives data from the middleware and uses it to control the movement of in game character.
- Kinect v2 (Xbox One): This motion-capture camera was used to detect the skeleton of a person. It uses a combination of cameras and infrared sensors to track the movements of a person's body. The SDK (software development kit) was also used to get data from the camera.
- Kinect Azure: Like the Kinect v2, this motion-sensing device was also used to detect the skeleton of a person. It the third generation of the Kinect cameras the Kinect v1 (Xbox 360) and the Kinect v2 (Kinect One), with higher resolution cameras and more powerful processing capabilities. Here I also use the provided SDK to communicate with the device, unfortunately the C# is not as well documented as C or C++ so it wasn't such an easy task.
- Existing projects: I developed all the features on top of the project I was working on the last semester, but I also reused code of the original project as the foundation for adding features like communicating with the web server.

5. DEVELOPED COMPETENCES AND SKILLS

Explanation on how the tasks performed have helped the student in developing the competences and skills defined in Annex I.

Developing this application and working on the tasks involved in this project has helped me in several ways by allowing me to develop a range of competences and skills. Some examples of these include:

- Programming skills: After a year of working with C#, I feel much more comfortable programming in this language, but I also see many more mistakes I made while creating this application and I am aware of them. Therefore, I had to learn how to reconcile the constant improvement of the code with adding new functionalities.
- Architecture skills: I learned a lot about the architecture of this type of systems, it is very important knowledge for me because it is not related to a specific programming language but is useful in all programming languages.
- Reconciling performance with readability: I've learned that sometimes it's worth giving up striving for the greatest efficiency in favour of readability and simplicity in further development. For this reason, I abandoned the old encoding system in favour of JSON which did not slow down our application and became much more readable.
- Manage time and task: Although I would like to refactor the code throughout the whole semester to make it perfectly readable, I had to learn to reconcile it with adding new functionalities and developing this application.

6. CONCLUSIONS

Evaluation on how the project has contributed to your training, and how the supervisor has influenced this process.

Since I started working on this project last semester, it has grown and become very complex. Therefore, it was very important to focus on the application architecture itself so that it would not get out of control. I hope I was able to learn a lot of good techniques and implement them in this project. I also had the opportunity to increase my knowledge of the C# language itself and object-oriented programming.

It was planned that during this project I would work with another student, which finally did not happen, but it allowed me to expand my knowledge about the use of git and GitHub platform, which I had the opportunity to use this semester.

7. PROJECT LOG

List and brief description of the weekly tasks in chronological order.

Schedule of activities from the beginning of the project:

7 weeks:

During this first period, I dedicated my effort to enhance the process of exchanging data between middleware and Unity by removing old format for JSON as a more modern format for data. I had to make this change both for middleware as well as for the Unity project. However not only was Unity had implemented JSON support during this period, but also most of its code was rewritten to improve clarity and readability. Additionally, I fixed middleware bugs that caused various kinds of errors, the biggest of which was Kinect stuttering.

3 weeks:

Over the course of next three weeks, I shifted my focus towards refactoring the code responsible for the user interface (UI). As a part of this task, I divided the main window file (MainWindow.xaml) into main, camera, log, and settings view. In the following weeks, login view was also added to this group. Moreover, I centralized the initialization of all globally accessed classes into a single location, specifically within the App.xaml.cs file.

1 week:

Within this one-week timeframe, I manage to implement an application localization feature, which implementation consists of two files storing key-value pairs for two languages namely English and Spanish. This implementation allowed for the seamless utilization of these languages both in user interface and in the application code.

1 week

Throughout a week I develop a code responsible for displaying a skeleton overlay over the camera image. This provides the user with real-time information about what exactly is detected so the user can address any undesired detections, ensuring a more accurate and reliable tracking experience.

3 weeks

During a span of last three weeks, I dedicated my attention to incorporating the logic responsible for the exchange of information between the middleware server and Unity. Middleware retrieves information from the server using the REST API regarding the exercises that the player will do in a Unity, then this information is sent to Unity using UDP. After completing the given exercise, Unity sends results of the exercise to middleware which will send it to the server at the appropriate time.

There were also plans for full integration the new system with "Phiby's Adventures". However, due to the complexity of the project, I was unable to achieve it. In place of that created the "Sample Unity Project for K2UM" this Unity project includes basic integration with the new K2UM.