Université de Strasbourg

NX3D

Internship Report

CSMI master degree 2020-2021

Development of a virtual reality application for real estate

University of Strasbourg

Mathematics and computer science department

Celia AZZOUG

Supervised by

Mr. Florian CAUTILLO

April 2021 - September 2021

Abstract

This internship realized in the company NX3D of Strasbourg concerns the development of a prototype of immersive virtual visit in 3D for the real estate promotion, and the creation of new functionalities in research and development, in order to improve the application which allows the users to navigate, to better visualize and to conceive their projects.

Therefore, it was necessary to re-study the application, from the data that uses to the smallest actions that performs, and to understand the general principle that was conceptualized, so that it could be subsequently modifiable and easily scalable.

The report is structured as follows: the context of the internship, the identified objectives and the presentation of the company are the subject of the first section. In the second section, we will present virtual reality and its history in general and in real estate in particular.

An overview of the main tools used and the organization of the internship are introduced in the state of the art in sections 3 and 4. The NX3D application and the knowledge models developed are described in section 5. The whole implementation part is detailed in section 6, from the architecture of the prototype of the dissemination system. The conclusion, section 7, closes the main part of the thesis. Annex sections are given to complete the information, both technical and associated with the application.

Acknowledgements

I would like to thank, first of all, my tutor Florian Cautillo for his availability, his advices and for having guided me in the different steps of my internship, and all the NX3D and ORIGIN team who facilitated my integration and for their welcome.
I would also like to thank Christophe Prud'homme for his help in the search for an internship and a big thank you to all the professors of CSMI, who provided me with the necessary tools.
Finally, I do not forget my dear parents, for their constant support and encouragement.

Contents

Appendices

1 Introduction

1.1 Context

In a world where technology is advancing at a rapid pace, virtual reality is improving and innovating every day. Today, virtual reality is making its mark with effective tools in various fields, and it has become a radical marketing and communication tool for the real estate sector.

Whether it's for entertainment or for touring homes, the technology and operation remain the same. However, to make a virtual tour of a property possible, it is necessary to model the property in 3D, either from plans processed by software or by using 360° photos of the property.

So, if buying a new property has many advantages, this purchase has a major problem on the buyer's side: you don't really know what you are buying. Future buyers are forced to visualize their future projects often on a 2D plan, with virtual images of the future property and photos of the plot. The virtual visit has much more impact, it allows to see all the corners of the property and to move in order to apprehend the various spaces. It allows future buyers to imagine themselves in the property.

Thus, Nx3D has chosen virtual staging technologies for the development of its application which has the ability to benefit both potential buyers and developers, therefore, real estate players who want to stay on the cutting edge of technology should consider deploying a virtual reality experience.

1.2 The company

NX3D was founded in February 2019 by Mr.Florian CAUTILLO and Mr.Frédéric PEREIRA, its headquarters is located at 5 Rue Kirstein - 67000 STRASBOURG.

The NX3D team is composed of :

- Florian Cautillo : CEO and co-founder of NX3D.
- Frédéric PEREIRA : Co-founder of NX3D and art director.
- Alain RICHOUX : 3D generalist in NX3D.

The company has developed a virtual tour tool at the request of its customers. It is an effective communication and marketing software for the real estate sector, promoters, builders and prospects, in order to allow them to give life to their architectural plans and offer them a realistic projection for a real-time visit integrated into the environment with various features such as:

- Discovery of the residence or the house: integrated in its natural environment (animation).
- Discovery of the volumes and the layout of the spaces: views, light, path and layout for each floor, animation and decoration through the configurator.

- Immersive visit of the home.
- Extraction: 3D animation, 360° immersive visit, orbital model, 2D sales plan, ...

1.3 NX3D's framework

The NX3D framework Figure 1 is composed of two parts: an API (SCOTTY) and the application (STORE).

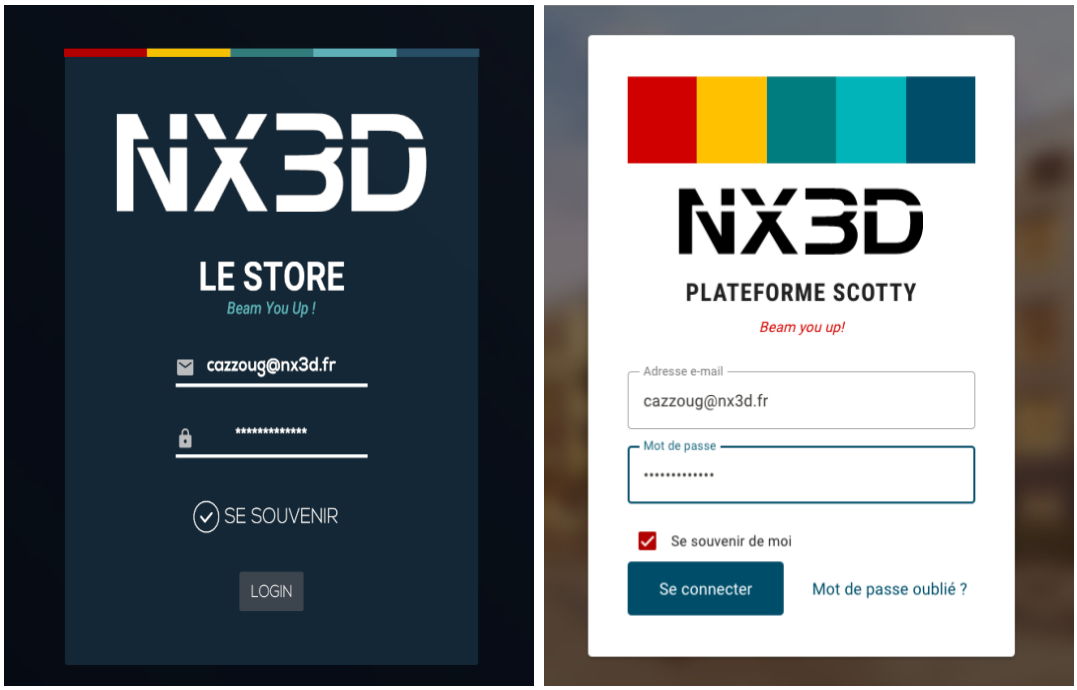


Figure 1: Scotty platform and Store of NX3D

`SCOTTY` [NX3D, 2019] is a web interface of the database on which the NXStore software is connected to access conversions and collaborations.

`STORE` is the application developed with Unity C# where the development part is composed of three main modules :

1. **UpdateCenter** is a set of configurable scripts, it handles all updates and patches for the Scotty platform and application modules, which are four main scripts :
   - `UpdaterProcessor` : It's the script that takes care of reading and writing the data in json format that is retrieved from the platform as well as verifying, adding and deleting the data
   - `UpdaterDatas`: Groups the functions of the updates of all the data(organization , agency , program , lot) Figure 2.

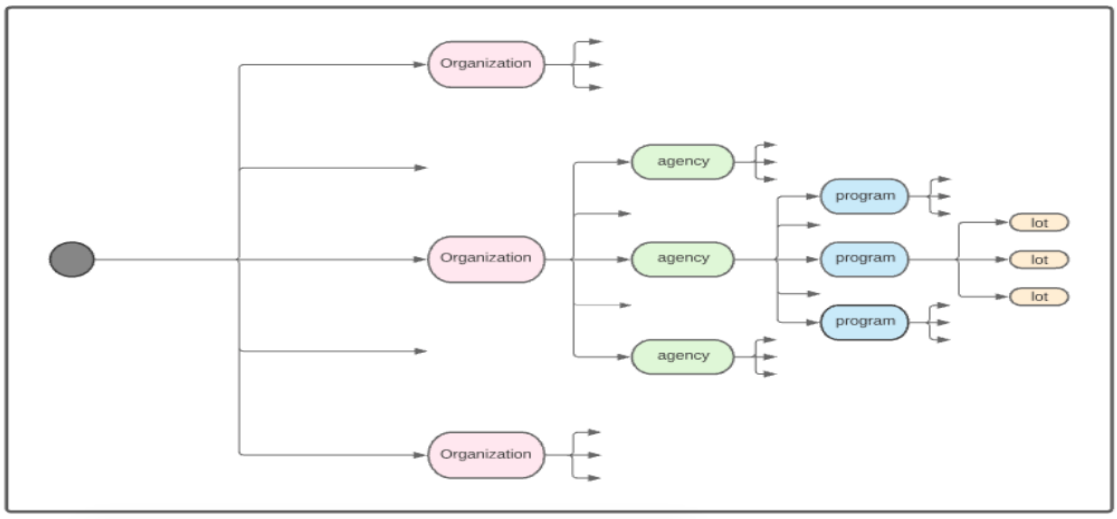


Figure 2: Data

- `UpdaterLibrairies`: It is about loading libraries(categories , items, ...) and their updates.
- `UpdaterBase`: Functions for creating, deleting and refreshing local and remote folders and files.

2. **Selection**: Groups the scripts of the different selectable objects of the scene which are the buildings, the floors and the houses , and which handles and manages them (add, remove, check type, enable..).

3. **CreativeManager**: It is composed of different types of scripts:

   - The login script ,managing cameras and player information.
   - Menu editing.
   - UI data management files (instantiations, accesses, ...) and creative UI managers that contain the different functions (access to visits according to the visit mode, housing reservation, localization, decoration, ...).
   - Ui Filters.

1.4 Project goals

The objective of this internship is to meet the different needs of NX3D by developing new versions of the application(Store), and to improve it by adding new functionalities to help future customers (buyers,...) and real estate professionals through this application to visualize or understand precisely a project, even before its construction with a less complex handling.

For this, it was necessary to have a knowledge environment that represents the different skills of the customer's choices according to his desires and needs, because it is important that

when he plans, he has the relevant information. The comprehension of this environment was the first step of the internship. To do this, an ontology will be built for each knowledge domain and each new feature to be developed, data will be elicited and local ontologies will be defined from these specific data.

The functionalities to be added to the application were defined at the beginning of the internship as follows:

- Addition of an access token.
- Detection of floors for multiplex housing.
- Reservation of housing.
- Make corrections on the mapbox.

Moreover, each task done has to be validated by my supervisor and then by the person in charge of marketing and where we discuss about probable improvements and correction of other features.

2 Virtual Reality

2.1 Introduction of virtual reality

According to the reference [Yaodong Sun, 2018] "Virtual reality (VR) technology is an integrated computer technology that can give the user a rapidly developing immersive experience. By creating a virtual scene, VR technology provides the user with an audio-visual or even tactile experience, and responds to the user's manipulation to give a sense of immersion.

The characteristics of the VR technique can be summarized as "three I's" Figure 3 which is **I**mmersion, **I**nteraction and **I**magination[Gang,]. In addition, it can also reflect people's subjective initiative. Based on the above characteristics, VR technique has been fully applied in different kinds of fields. For example, NASA has developed a training system for aviation and satellite preservation, SRI international conducts aircraft or vehicle navigation training via VR techniques [Zhonghua, 2004], in order to reduce the probability of flight or driving accident.

VR technique has a great influence on fields that emphasize hands-on approach or those with low error tolerance, such as medical engineering and medical education, especially surgical therapy whose VR techniques provide a multiple and repetitive chance for students to practice their maneuver".

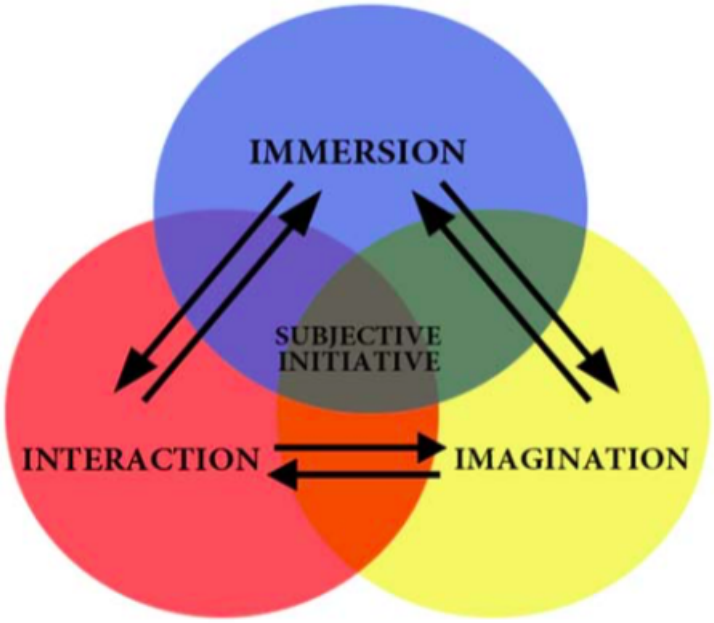


Figure 3: Virtual Reality Triangle [Yaodong Sun, 2018]

2.2 History

It seems important to start with a little history of virtual reality and know where it comes from. Computer researcher Jaron Lanier introduced the term "virtual reality" in 1985 [UQO,].

As for the concept, it dates from long before the creation of computers, Platon is the first to describe virtual reality in his book "The Allegory of the Cave", such that the people attached can only see their own projection and those of others in the form of shadows projected on a wall thanks to a fire. They hear only the echoes produced by the walls of the cave, their auditory and visual senses are biased by an immersive and sensory altering device [Platon, vJ C].

In a passage of the "Discourse of the method" in which Descartes puts in doubt the reality which considers it to be only impressions sent by a demon (deceitful spirit)[R., 1637] and in the "Principles of the human knowledge" Berkeley exposes his theory of the knowledge and his immaterialism, summarized by the sentence, "to be, it is necessary to be perceived" [G., 1710].

For Berkeley, it is a perception of ideas and nothing can be conceived outside of them. Hence the non-assertion of the existence of the external world, because we only perceive it through the idea that we have of it. These two writings show a virtual reality, which is perceived only through our senses and the analysis that follows from them [G., 1710].

It will thus be necessary to wait until 1962 for the installation of systems of virtual reality, with the invention of the sensorama Figure 4 by Morton Heilig which allows the diffusion of a stereoscopic image with a sound, but also odors and wind at the time of a film diffused for a spectator alone.

Figure 4: Sensorama

It was only in the 90s that the first devices as we know them today were prototyped, and it was not until 2012 that Palmer Lucky announced the development of the Oculus Rift Figure 5, which guarantees "immersion" and "navigation" in the image since it is directed by the eyes, exactly as in reality [Andre, 2017].

Figure 5: Oculus Rift

2.3 Virtual reality at the disposal of the real estate industry

For most people, virtual reality is for entertainment and associated with video games, but what some people don't know is that this technology has huge potential in the real estate

industry.

We've all been through the process of buying or renting a property and we all know how that process works. Typically, a real estate agent provides a long list of properties to the client, followed by explanations, information, negotiations and, finally, actual visits of houses and apartments. This workflow has remained the same for a long time, although it is actually inconvenient and time-consuming and can even be costly, both for real estate agents and buyers.

And to alleviate these problems virtual reality technology is being used, as it allows millions of people to virtually visit properties without moving. And it offers a total immersion experience and that requires using a virtual reality headset to experience immersive three-dimensional tours. In a short time, potential buyers or tenants can virtually visit several locations and decide which ones deserve an in-person visit.

As for real estate companies, virtual tours allow them to market properties with very little investment and to market the finished project before construction is even completed [B., 2020].

3 Implementation tools and protocols

In view of the objectives to be reached, it was important to choose the elements necessary to carry out these various tasks.

The first tool used is the game engine `Unity` associated with the IDE (Integrated Development Environment) `Visual Studio` as development environment. Unity is generally used for the creation of video games, but its possibilities in terms of 3D rendering, application of textures and animations made it an ideal tool for the realization of the application. Visual Studio is used to produce the source code of the new functionalities in C# language, supported by the game engine. The two programs work in association, which makes it possible to directly observe the visual changes in Unity when modifications are made in the source code with Visual Studio.

The application is based on architectural plans, but to integrate the files on Unity, we first model the 2D plans in 3D on `ArchiCAD` and thanks to the `Twinmotion` add-on module of `ArchiCAD` the 3D models will be exported in `.FBX`(Filmbox File) format to Unity where they will be used.In addition, for data retrieval, we use `UnityWebRequest` based on http protocol is was used.

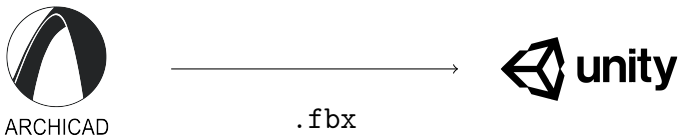


Figure 6: Integration process

Finally, to keep an archive of the project and to have the possibility that it has several speakers in the development, the use of a server `Git` was essential with the graphic interface `Git Extensions`. This tool allows to share the source code to several people and to have a history of versions.

3.1 Unity 3D

Unity3D is a software for creating virtual environments that has a physics engine based on the PhysX engine from Nvidia. It has several advantages, first of all, its library is quite extensive and complete for mathematics, physics and camera effects. Its drag-and-drop interface system is also very intuitive. The professional version also has a shadow engine and an excellent water rendering. Programming is essential: under Unity, everything is scriptable, due to the C# language [NOVAE, 2019].

To form an application, you just need to assemble elements in the editor through dependencies and interactions under the name of Assets. This can be descriptions, sounds,images or 3D models. Some of them are already included in Unity3D, especially physical interactions. Moreover, we can also use different packages and resources external to the software to realize the application that we can import from the `Unity Asset Store` and that correspond to a set of assets developed by the Unity community, which can be reused in the projects, this editor is provided with the development software `MonoDevelop` in order to be able to realize the scripts.

Finally, this editor also has a `GameObject` system that are all the objects present in a scene. They constitute a kind of container in a Unity scene. They are associated with various components such as meshes, lights, scripts, materials, ..., they can thus represent a playable character, a set element, a camera, a light, ...

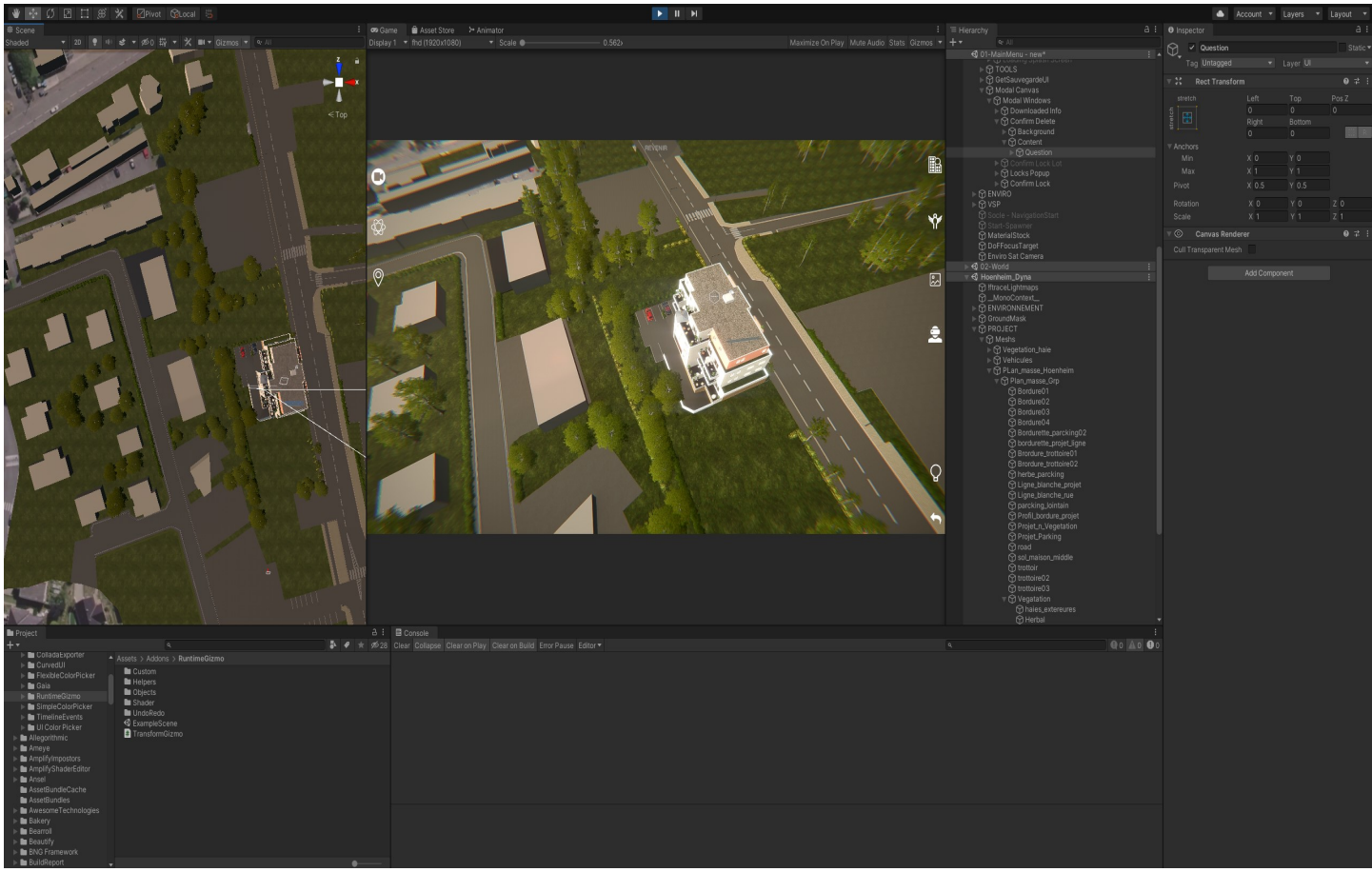


Figure 7: Unity3D interface

3.2 ArchiCAD

Published by Graphisoft, ArchiCADis a 3D architectural design software of BIM (Building Information Modeling) type.

It is dedicated to the design of documents used by architects (computer graphics, quantity takeoffs, perspectives, plans, ...).

The BIM concept consists in building a 3D digital model from plan, section, elevation, axonometry or perspective views. The edition of an element (land, wall, roof, door, window ...) in one of these views is automatically reflected in all other views.

It integrates drawing tools in two dimensions and tools for creating and modeling architectural elements in three dimensions, which allows you to make meshes, superimpose floors, ...[Informatique,]

3.3 UnityWebRequest

HTTP is a communication protocol for transferring documents between a client (machine sending requests) and a server (machine responding to these requests). This protocol is used by the Web servers hosting Internet sites, with the aim of allowing to download documents as well as the consultation of pages on the screen of the client [Contributors, 2021].

The UnityWebRequests manage the HTTP communication flow with the web servers. Other objects in particular DownloadHandler and UploadHandler handle the downloading and uploading of data respectively [Unity, c].

A set of static functions are provided, which return properly configured UnityWebRequest objects for many uses.

HTTP defines a set of request methods that indicate the action to be performed on the specified resource, such as [Nandaa,]:

- `GET` requests a representation of the specified resource. GET requests should only be used to retrieve data.

  The result of GET requests will be in one of the following formats: either a JSON, an image, or an error.

  JSON (JavaScript Object Notation) is a lightweight writing format for data transfer. Thus to recover the various data it is necessary to "deserialize" the JSON. It will then be in the format of a dictionary.

- `POST` is used to send an entity to the indicated resource. This usually causes side effects on the server or a change of state.

- `PUT` replaces all current representations of the target resource with the contents of the request.

4 Organisation

4.1 Work method & Communication

Florian Cautillo regularly gave me new objectives as the tasks were being carried out. After a dialogue aimed at refining the different aspects of a new function to be integrated, I designed the task to be accomplished, then I proceeded with its development. Once the functions were added to the application, I corrected any bugs discovered during programming and verified that the other related functions were still working.

And as far as communication is concerned, the members of the project team being present in the company, verbal communication is the preferred means of communication. Since the team is very available, it is easy to get quick answers to specific questions as well as to get interviews or even meetings. Otherwise, it was possible for me to request information from the entire team via Slack, for simple questions that needed a quick answer, such as questions about certain details of the application already in place or procedural points in order management. When the people involved were not present, they could be called by phone.

4.2 Materials used

Working with two screens, the first one was dedicated to the Unity software which allowed me to see the repercussions of the modifications made in the source code, containing, among other things, a view of the 3D scene, and the second one was used to visualize Visual Studio from which I could modify the source code of the application.

5 Study of the existing

Before working on the different objectives, I first had to understand the software architecture set up by the previous groups. So I started by exploring all the scripts of the Unity project in a general way to better understand and notice the structure of the source code, I set up a summary of process execution.

By default, a script inherits from the MonoBehaviour class

- which inherits from Behaviour (any component that can be enabled or disabled).
- which inherits from Component (anything that can be attached to a GameObject).

```
                    Default C# script

using UnityEngine;
using System.Collections;

public class #SCRIPTNAME# : MonoBehaviour {

    void Start () {   // Use this for initialization
    }
    void Awake () { // Is called after the Start functions.
    }
    void Update () { // Update is called once per frame
    }
}
```

Scripts govern the behavior of GameObjects. In the case of Unity for the creation of an application for example, we don't need to create the code that runs the application, because Unity does it for us for example either by attaching scripts to the gameObjects or through functions that run automatically in Unity such as [Unity, a]:

- `Awake()` is called only when the GameObject element with this component is initialized. If a GameObject is inactive, it will not be called until it becomes active. However, Awake

is called even if the GameObject is active but the component is not activated. Awake can be used to initialize all variables that you need to assign a value to.

- `Start()` like Awake, Start will be called if a GameObject is active, but only if the component is activated.
- `Update()` is called once per frame, this is where we place the code that defines the logic that runs continuously, such as animations and other parts of the game that need to be updated continuously.

Process execution:

```
using System.Collections;
using CreativeManager;

public class UpdateCenter : MonoBehaviour {
    private readonly List<UpdaterBase> m_Updaters = new List<
      UpdaterBase> ( );
    public string m_RootURL = "https://cloud.nx3d.fr/";
    public string m_StagingRootURL = "https://staging.nx3d.fr/";
    public ConnexionManager m_connexionManager;

    private void Start ( ) {
        if ( m_initOnConnection && m_connexionManager != null )
          {
            m_connexionManager.OnConnected.AddListener (
              delegate {
                Init ( );
            } );
        }
    }
```

The above code is executed when a user connects to the application, where the `Start` function is automatically executed at the connection.

```
public void Init ( ) {
    PlayerInfo mainPlayer = null;
    if ( m_playerInfoManager != null ) {
        mainPlayer = m_playerInfoManager.MainPlayer;
    }
    if ( m_updateLibrary ) {
```

```
        AddUpdater ( new UpdaterLibrary (m_RootURL ,
          m_SrvLibraryURL , mainPlayer , localBibPath ,
          remoteBibPath , commonLocalBibPath );
    }
    if ( m_updateDatas ) {
        AddUpdater ( new UpdaterDatas (  m_RootURL ,
          m_SrvDatasURL , mainPlayer , m_updateFurnitures ) );
    }

    StartCoroutine ( CheckForUpdate ( ) );
    }
```

The function `init()` is called in `Start()` so that it is executed at the start of the application, this function contains two functions `AddUpdater()` which launches the function `UpdaterDatas()` in the case of data update and `UpdaterLibrary()` for updating libraries, and `CheckForUpdate()` allows a refresh every precise time and this one is called with a `StartCoroutine()` which means an execution in parallel.

Once a user is logged in, a folder named with the user's email (e.g. cazzoug@nx3d.fr) and a `common` folder are created .

- The folder `"cazzoug@nx3d.fr"` stores all the informations of the concerned user and all the data of the programs as well as a subfolder `"srvData"` which saves the server data which lists two files: the local and the remote one, the remote one stores the data downloaded remotely (from scotty) which is updated at each connection or refresh contrary to the local one which is intended for the data managed by the application.
- As for the `common` folder, it archives all data that are common between users such as libraries.

Figure 8: Visit mode

Thus during an open session we have two choices of visit mode as we can see it in the diagram above Figure 8 either to make a direct visit of a housing Figure 9 or rather to opt for a visit of program Figure 10 which consists first in choosing the program then in selecting the building then the floor and finally the housing which one wishes to visit.

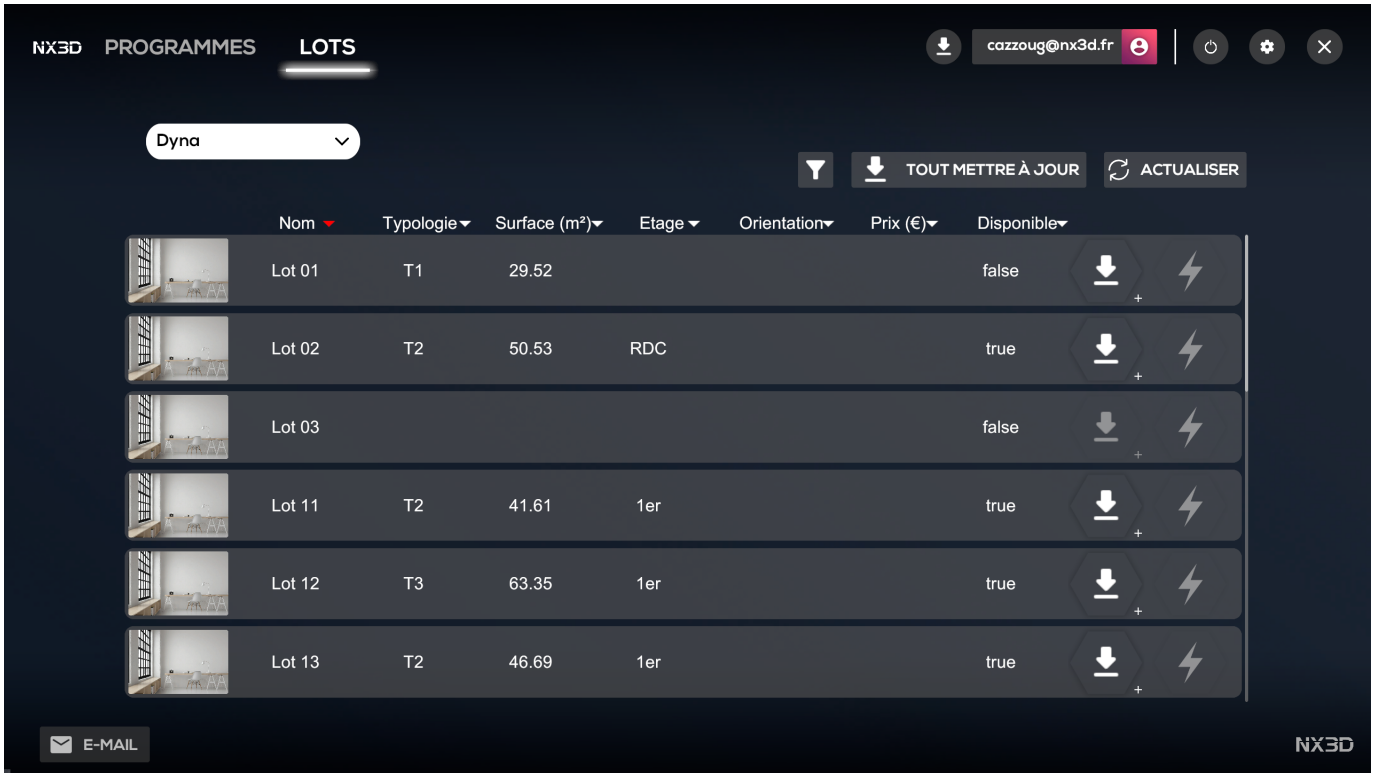


Figure 9: This figure represents the direct view mode of a housing, in this mode we can see that for each selected program we list the housings that it contains, and for each housing we find all the information about it and two buttons, namely the download button and the launch button (visit), as well as the possibility to order the housings according to an attribute (name, typology, etc.). We also have the main buttons, which are the update button, the filter button and the refresh button.

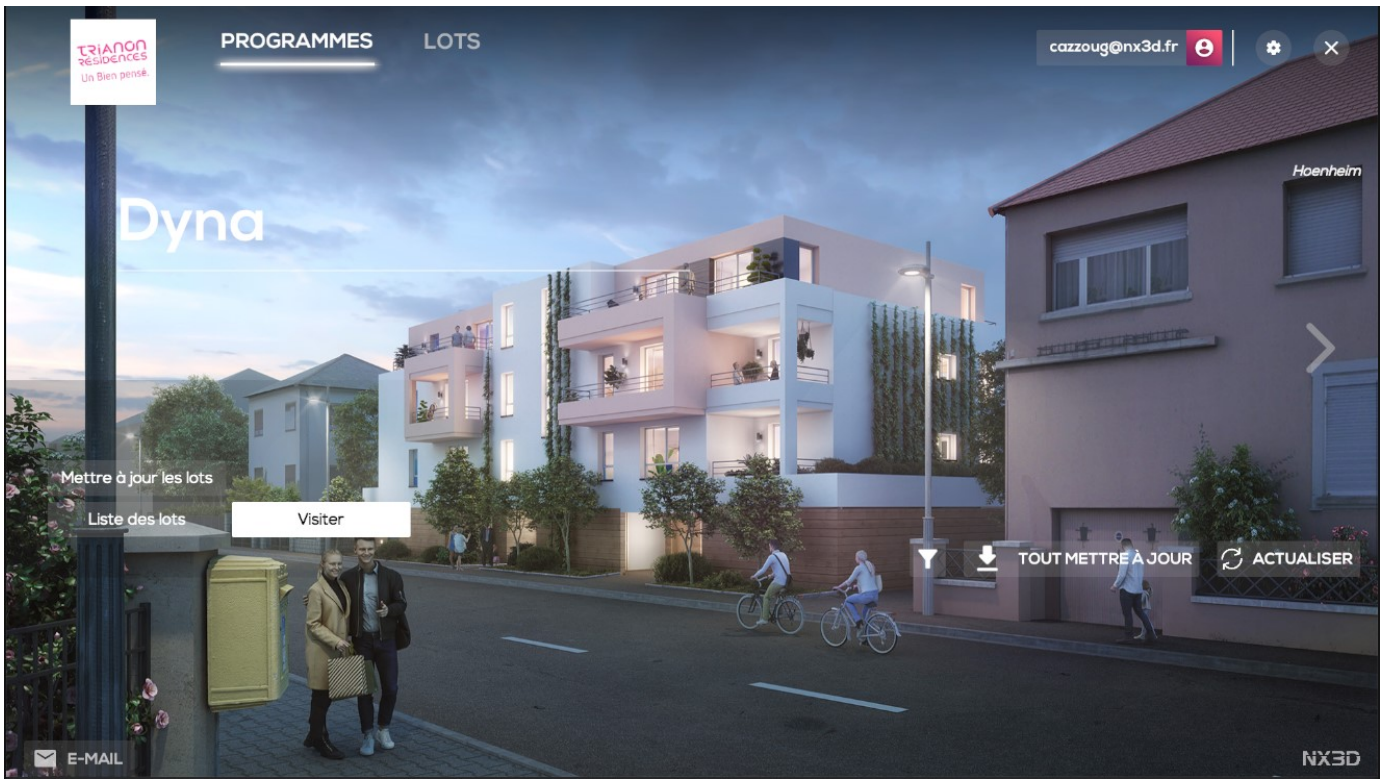


Figure 10: This figure shows us another mode of visit which is visit of the program, for the choice of the program we can slide to the left or to the right to select the program, as the first mode we have buttons of: loading, filtering, updating, refreshing and downloading the program, and we also find its information such as the name of the program and its address .

6 Implementation of the project

For the implementation of the project, I chose to follow a three-step development process to meet each objective. The implementation of these objectives started each time with a specification phase with my internship supervisor, then continued with a design phase covering multiple aspects and ended with a development phase. We will also see the results obtained from this process.

6.1 Access Token

In order for the server to respond much faster to authentication requests, no matter how many users are connected, to provide an additional level of security for accessing data and to avoid session disconnection problems, we retrieve a piece of data called `xAuthToken` which is an Access Token.

The Access Token is a string of characters that corresponds to the way an application identifies a user. When the user calls to connect to the application, an Access Token is given to him in accordance with the authorizations he has associated to the application.

To retrieve this data we have for each user a JWT token, valid for 7 days, which should

be used as long as the user is connected to the application store, and to be able to use it, we must put it in the header of all HTTP requests.

And as at the connection we already retrieve the data including the user's information and this Token, however, we only needed to declare the variable `xAuthToken` which is a string, add it to the class and then modify the function that reads the retrieved data to retrieve this value so it can be used.

6.2 Floor detection for multiplexes

For this new functionality the goal is to be able to visualize the floors of a multiplex and their detections being selectable objects in the program visit mode.

Before looking at the way I implemented this functionality,we saw in section 5 that the structure of a Unity program is made of game elements that can contain components. The first step of this task was the retrieval of the data from the user selected dwelling, which can be summarised in the following code.

```
public void GetLevelsForLot(){
    GameObject LevelToAdd;
    for (int i = 0; i < targetParent[1].transform.childCount; i
      ++)
    {
        if ((targetParent[1].transform..GetChild(i).name.Tolower
          ().StartsWith("niveau")) || (targetParent[1].transform
          .GetChild(i).name.Tolower().StartsWith("rdc")))
        {
            LevelToAdd = targetParent[1].transform.GetChild(i).
              gameObject;
            //attach the script. SelectableLevel to the
              gameobject
            SelectableLevel sel = LevelToAdd.AddComponent<
              SelectableLevel>();
            sel.Object = new LevelObject(LevelToAdd);
            sel.enabled = false;
            m_LevelSelectablesforLot.Add(sel);
        }
    }
    IdentifyLevelIndex(m_LevelSelectablesforLot);
    m_LevelSelectablesforLot = m_LevelSelectablesforLot.OrderBy(
      level => level.Object.Index).ToList();
}
```

`GetLevelsForLot()` : summarizes how the detection of the floor was done, by exploiting the data of the . fbx generated, we have in the mesh of each lot the objects which compose it, including its floors, and as the floors are distinguished by its names: rdc or level 0 for the ground floor and level (+number) for the upper levels, then for each floor detected it was necessary to attach to it the component `selectableLevel`(item 2) to confer the behaviour of an object of the type floor.

Moreover `GetLevelsForLot()` contains the function `IdentifyLevelIndex()` which identifies the index of the floor to determine the lower and upper floors and this is based on a comparison of the position of the y-coordinate so that at the end they are ordered according to their indices.

The next step was to integrate this function into the application. To do this, we had to recreate selectable buttons for the floors, adapting it to the application's interface.

However, for each visited lot we browse its list of floors returned by the `GetLevelsForLot()` function and depending on the number of floors we generate selectable buttons.

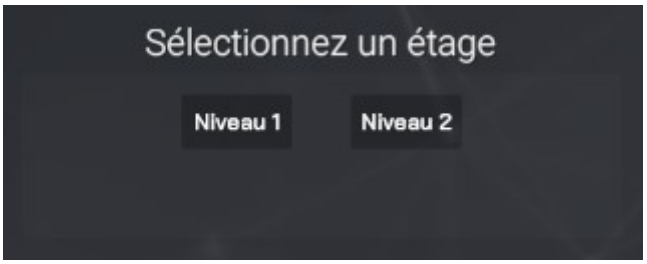


Figure 11: Selectable levels for lot

Once the clicks on the floors are generated, it is necessary to integrate a function that controls the display of the selected floor at each click, and to develop this I exploited what has been done in the concept of the building floors, by reusing and modifying it I managed to display the floors, then to have a centered view in relation to the camera I had to find a way to detect the center of each floor, and as each project contains a mesh, I based on the rendering of the boundaries, ie the boundary box aligned on the axis that completely surrounds the object in space, and since each floor has a Slab object that delimits it, then it was the favorable object to find a good center.

I also made sure that when we press a floor button, we are in an orbital view and the player's position is updated so that once we are in an FPS view (immersive view) the navmeshagent (this is the component that is attached to a mobile character in the game to allow him to navigate the scene using the NavMesh) moves to the selected floor.

```
Onselected() {
    DisableUpperLevels();
```

```
    SetMenuHomeDetails(selectableLevel,CurrentSelectableLot.Lot
      );
    //detect center of level
    Renderer[] renderers = selectableLevel.
      GetComponentsInChildren<Renderer>();
    foreach( Renderer renderer in renderers){
        if (renderer.name.StartsWith("Dalle")){
            Vector3 newCenter = renderer.bounds.center;
            //update player and camera position
            SetFPSCameraMode(newCenter, ZoomForLot(
              CurrentSelectableLot) / 2);
        }
    }
    //mode Orbit
    SetOrbitalCameraMode(ZoomForLot(CurrentSelectableLot) / 2);
    OnOrbitMode();
    //zoom in on the center
    UpdatePlayerTransform(true,CurrentSelectableLot.Center,
      ZoomForLot(CurrentSelectableLot) / 2);
}
```

`Onselected()` is the function called when a floor is clicked, it contains the function `DisableUpperLevels()` which disables the upper floors for the selected floor and the function `SetMenuHomeDetails()` which generates the selectable floors.

But I had a problem, is that when I click on a floor, the navmeshagent does not move, to mitigate this problem I made sure to disable the navmeshagent before moving it and reactivate it right after because it will be difficult to move it without following the path (for example if there are stairs as in this case it takes the stairs to teleport).

After some tests I noticed that it takes a double click for the navmeshagent to teleport to the desired position, however I started to look for where it came from until I found a forum that quoted the fact that the navmeshagent has a time constraint to teleport, so I added a `wait()` function with a waiting time of `3ms` (minimum time for the navmeshAgent to teleport)

```
foreach (SelectableLevel selectableLevel in selectablesLevels)
{
    GetComponent<Button>().onClick.AddListener(delegate
    {
        m_navmeshAgent.enabled = false;
        OnSelected(selectableLevel);
        StartCoroutine( Wait(0.3f ));
```

```
    });
}
```

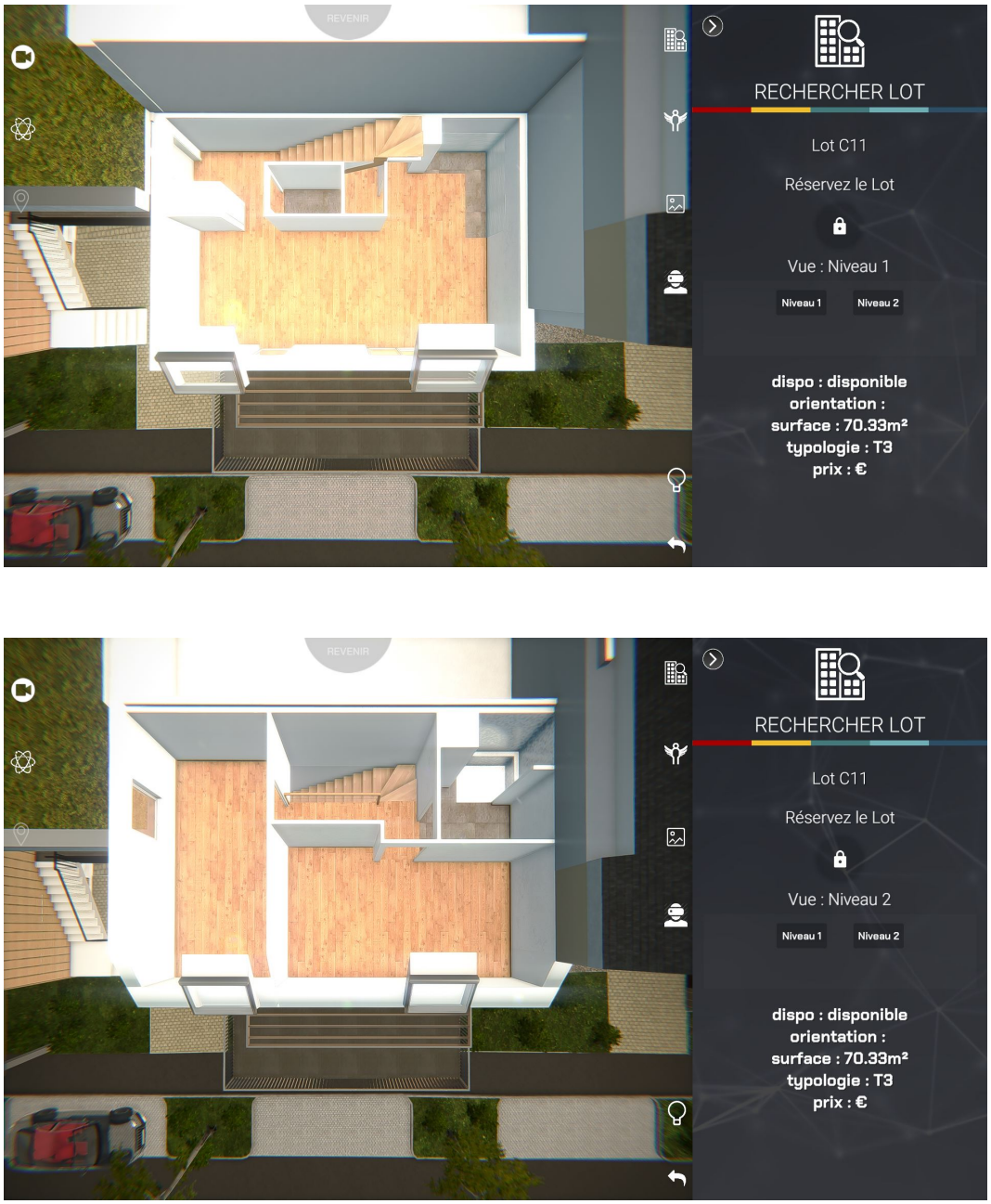


Figure 12: These two figures show the result obtained, it shows us the view on level1 and level2 of the lot

6.3 Reservation of lots

The purpose of this feature is to give customers the possibility to lock the accommodation they like. The first step was to obtain the list of customers and to set up a search system, then to make the various requests for locking the lot.

6.3.1 Client list

It should be noted that the reading of the data requires the creation of abstract classes according to the `.json` format of the retrieved data where it is necessary to name the variables in the same way as they are stored whence Figure 13.

```
public List<CustomerInfo>
  data = new List<
  CustomerInfo>();
[Serializable]
public class CustomerInfo
{
    public int id;
    public string firstname;
    public string lastname;
   }
```

```
"data":[{id : 1,
        firstname : "Azzoug"
          ,
        lastname : "Celia"},
        {id : 2,
        firstname :"Nadji",
        lastname : "Ferial"
          },
        {id : 3,
        firstname :"Cautillo
          ",
        lastname : "FLorian"
          }]
```

Figure 13: We can see in this figure on the left how the client class has been declared and on the right an example of a retrieved data

After declaring the class the next step is to read the information from the .json file. To do this we must first make a UnityWebRequest GET request (subsection 3.3) to retrieve the data.

```
UnityWebRequest webRequest = UnityWebRequest.Get(
  CreativeConstants.ROOT_URL + "/customers/index");
// Request and wait for the desired page.
yield return webRequest.SendWebRequest();
```

Once the data is retrieved, the response received from GET must be converted thanks to the `Newtonsoft.Json` package which offers with the `JsonSerializer` method, the possibility of converting a `JSON` text into a `.NET` object.

`JsonSerializer` converts `.NET` objects into their `JSON` equivalent and vice versa by matching the property names of `.NET` objects to `JSON` property names and copying the values, using the `SerializeObject()` and `DeserializeObject()` methods of `JsonConvert`.

```
UserSrvInfo customersInfo = (UserSrvInfo)Newtonsoft.Json.
  JsonConvert.DeserializeObject(webRequest.downloadHandler.text,
   typeof(UserSrvInfo));
```

```
if (customersInfo != null)
{
    if (customersInfo.data != null)
    {
        foreach (CustomerInfocustomer in customersInfo.data)
        {
            id = customer.id;
            firstname = customer.firstname;
            lastname = customer.lastname;
        }
    }
    else
    {
        customersInfo.data = new List<CustomerInfo>();
    }
  m_customers = customersInfo.data;
}
```

The list of clients is successfully retrieved, it is now necessary to define a search system.

In order to obtain results each time a search is performed, a search engine must be developed. So I created two new scripts `ClientFilter` and `AutoCompleteClientComboBox`.

- `ClientFilter` includes the function that initialize the client list.

```
public Dictionary<string, CLIENT_INFOS> m_allClients;
public void Init()
{   m_allClients = new Dictionary<string, CLIENT_INFOS>();
    m_customers = ConnexionManager.Instance.m_customers;
    for (int i = 0; i < m_customers.Count; i++)
    {
        CustomerInfo currentData = m_customers[i];
        CLIENT_INFOS newClient = new CLIENT_INFOS
        {   Prenom = currentData.lastname.Trim(),
            Nom = currentData.firstname.Trim(),
            id = currentData.id.ToString() };

        newClient.NomPrenom=newClient.Nom+" "+newClient.Prenom;
        newClient.PrenomNom=newClient.Prenom+" "+newClient.Nom;
        if (!ClientAlreadyExist(m_allClients, newClient))
        {
            m_allClients.Add(newClient.id, newClient);
```

```
        }
    }
}
```

- `AutoCompleteClientComboBox` contains the functions to create the search system including the Filter function which allows to filter the client list according to what has been typed by managing upper case, lower case, space, composition (name+firstname, firstname, id, ...) like this :

```
bool nameMatch = option.Nom.ToLower().Replace(" ", "").Replace("
  -", "").StartsWith(curText.ToLower().Replace(" ", "").Replace(
  "-", ""));
```

```
if (nameMatch || idMatch || prenomMatch || NomPrenomMatch ||
  nameAndIDMatch || PrenomNomMatch)
{
    filteredOptions.Add(option.ListeDisplay);
    filteredIds.Add(option.id);
}
```

So far, the only thing we have said about this is how to retrieve the list of customers and develop a search system, but this must be passed into the application, for which we pass the UI elements that are simply elements in the game (and most often, on the screen) that show useful information to the player, that are generally persistent in their presence.

However, the display and animation of the window lock the lot requires creating a canvas that acts as the master of all UI elements on the screen. Therefore, all UI elements must be child game objects of the Canvas game object. The Canvas [Unity, 2020] in Unity is an area in the game where you can draw UI elements. And for the client list, we need to create a `Customer` prefab Figure 14, but first we need to create a lock button that can be clicked to trigger the reservation event in the different access visits (program /lot).

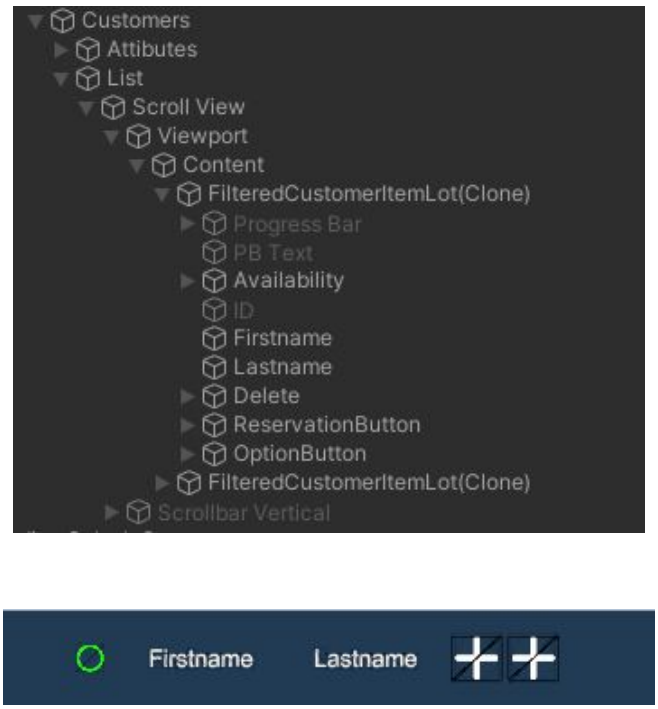


Figure 14: Customer prefab

Having two modes of visit, it was necessary to create a canvas for the visit program and another for the visit of batch, which will be adapted to the interface of the application, for the program mode Figure 15 I add the canvas as much as child in the interface of menu which will be posted with the click on the button lock, as for the mode visit lot Figure 16 I opted for an opening of a pop when one also clicks on the button Lock.

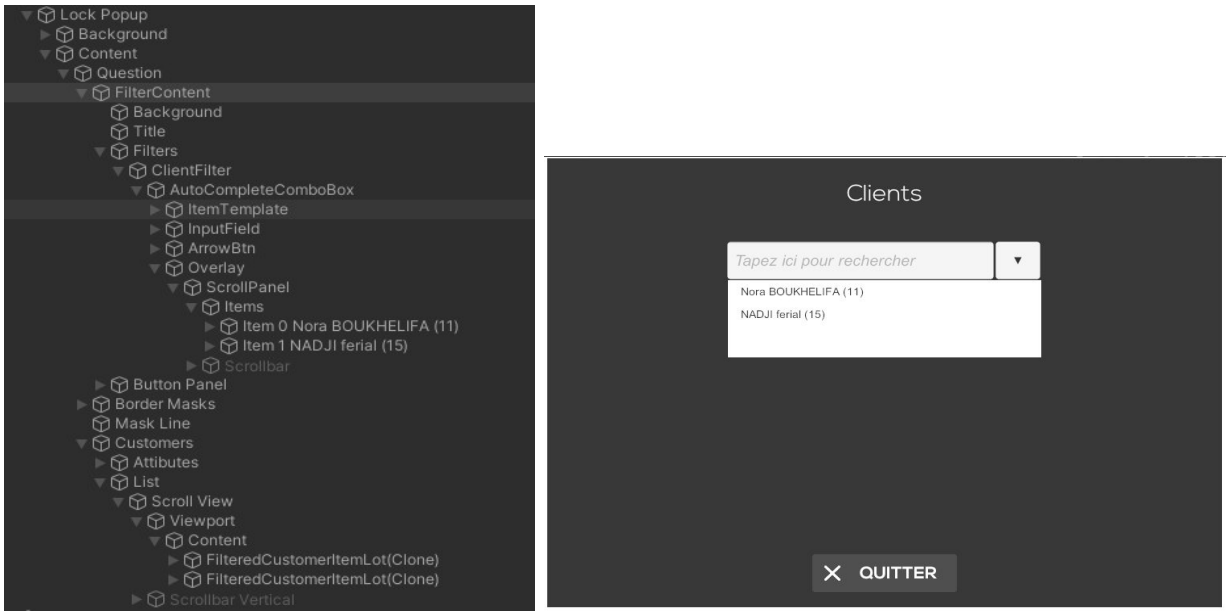


Figure 15: In this picture you can see on the left the hierarchy of the created canvas and on the right the Pop Up for the program visit mode.

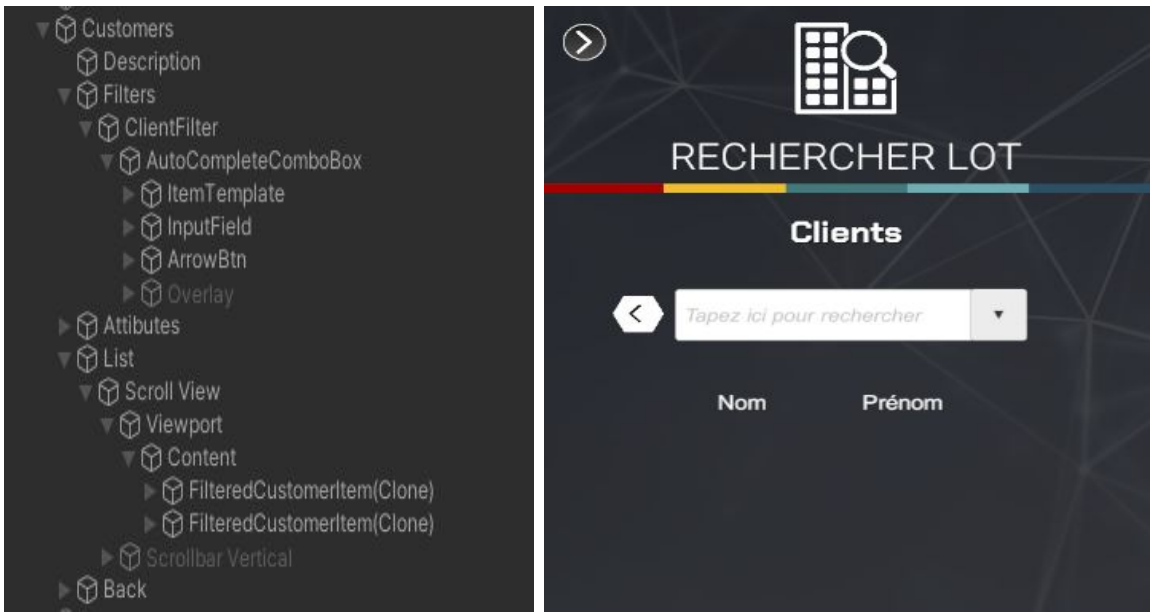


Figure 16: These two figures above, we have on the left the hierarchy of the canvas created in the menu of lot visit and on the right the rendering in the interface of the application.

6.3.2 Lock the lot

Once the list of clients is collected, it remains to define the locking of the lots, first of all we explain the functioning of the locking.

After a client has looked around the properties, there is one that catches his attention, so he has the possibility to make a reservation if he is convinced of his choice, but in case he wants to sign a reservation contract for the lot he wants to buy, he has to find out some information, nevertheless, he does not want the lot he wishes to buy to be sold during the time necessary for his investigations, so he can ask the seller for an option on the reservation of the lot he is interested in for a period of n days (the n is defined according to the organisation, otherwise by default = 3 days).

To summarize, we have two main choices: either to reserve the lot, or to make an option on the reservation, we also have other possibilities to extend the option and cancel the reservation. That requires to make requests POST (subsection 3.3) for each link and it will be necessary also to be connected (to have a `X-AUTH-TOKEN`(subsection 6.1)) thus it is necessary to put the token with the head of each request.

Lock Lot: this query allows the reservation or option of the lot.

```
public IEnumerator LockLotRequest ( int customerID, Lot lot ,
  string lockType)
{
    string url =ROOT_URL + "/customers/" + customerID + "/lock-
      lot/" + lot.id;
```

```
    Dictionary<string, string> content = new Dictionary<string,
      string> { { "Content-Type" , "application/json" }};
    content.Add("X-AUTH-TOKEN", xAuthToken);
    content.Add("lockType" , lockType);

    UnityWebRequest request = UnityWebRequest.Post(url, content)
      ;
    yield return request.SendWebRequest();

}
```

- `/customerID/lock-lot/lotID` : url for the lock.
- `customerID` is the customer identifier.
- `lotId` is the lot ID.

We have a parameter in the content of the request that follows the Access Token which is `lockType` it is a string that defines the type of lock.

- `lockType` : [option | reservation], if option, an option deadline will be added automatically to the database. It is calculated via the organisation's configuration.

Extend option Lot :it is used to extend the lock.

```
public IEnumerator ExtendLockLotRequest(int customerID, Lot lot
  , Organization org)
{
    string url =ROOT_URL + "/customers/" + customerID + "/extend
      -lock-lot/" + lot.id;

    if (org.option_delay != 0)
    {
        optionDelay = org.option_delay;
    }else // default val 3days
    {
        optionDelay = 3;
    }
    newOptionLimitDate = newOptionLimitDate.AddDays(optionDelay)
      ;
```

```
    content.Add("X-AUTH-TOKEN",xAuthToken);
    content.Add("new_option_limit_date", newOptionLimitDate.
      ToString("dd/MM/yyyy"));
    UnityWebRequest request = UnityWebRequest.Post(url, content)
      ;

    yield return request.SendWebRequest();
}
```

- `/customerID/extend-lock-lot/lotID`: url for the extension. It has one more parameter in the request content than the Access Token which is `newOptionLimitDate`
- `newOptionLimitDate` (string, format dd/mm/YYYY): New option limit date

Unlock Lot: this is the request to unlock the lot.

```
public IEnumerator UnlockLotRequest(int customerID, Lot lot)
{
    string url = ROOT_URL + "/customers/" + customerID + "/
      unlock-lot/" + lot.id;
     Dictionary<string, string> content = new Dictionary<string,
        string> {{ "Content-Type" , "application/json" } };

    content.Add("X-AUTH-TOKEN", ConnexionManager.Instance.
      xAuthToken);
    UnityWebRequest www = UnityWebRequest.Post(path, content);

}
```

- `/customerID/unlock-lot/lotID` : url for the unlock.

After the declaration of POST requests, the locking is accessible from a button that controls an animation of the opening and appearance of the different graphic elements, whether in lot visit mode or in program mode. Once the menu is visible, the customer search bar appears and each customer is displayed with his name, first name, the availability of the lot and two buttons representing respectively the reservation and the option of the lot.

But the task is not simpler than that because once a lot is reserved or an option is set, the data is not updated on the local file as it is re-explained in 5, for the state of the lot is

updated, it was necessary to write the new data in the local file, so when a client locks a lot (reservation/option) the new data will be stored and it will be possible to read it to update the availability and the user interface elements,

However I created a truncated reservation class (without taking into account all the information of a lock and the customer).

```
[Serializable]
public class reservation{
    public int cutomerId;
    public string lockType;
}
```

I developed a new function `UpdateStateLot()` which adds the new data, so that at each launch of the visit the data modified during a session will be added to the local file.

How does it work?

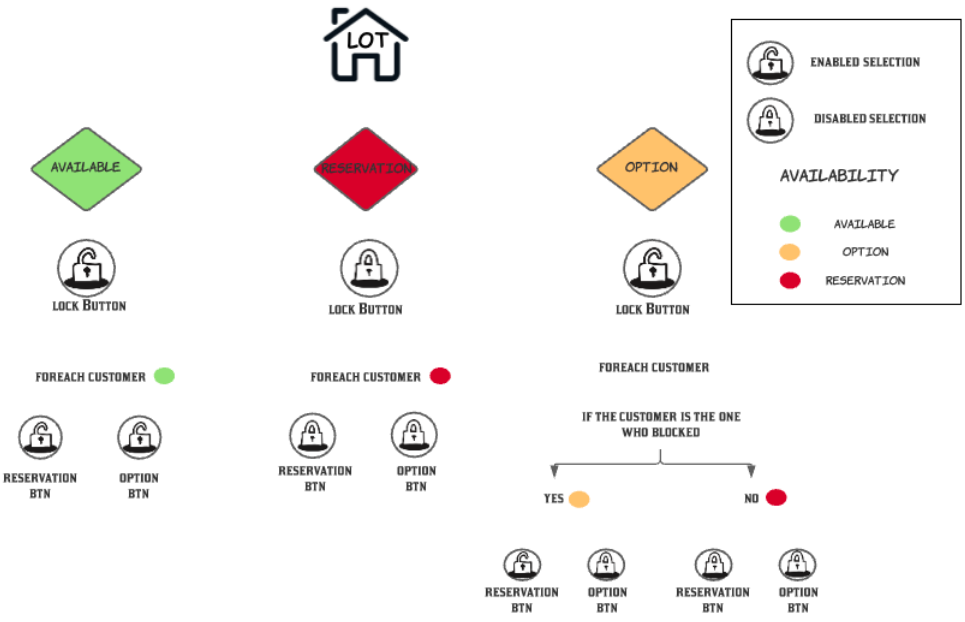


Figure 17: How the update works?

The figure Figure 17 summarizes the function `UpdateStateLot()` for each Lot "X", for each client, as indicated in the prefab Figure 14, there is one availability and two buttons (reservation/option), but there are several cases:

- Lot "X" reserved → lock button inactive, availability red and both choice lock buttons are inactive

- Lot "X" option → lock button enabled, orange availability, option button disables, reservation button active only for the person who made the option, otherwise disables.
- Lot "X" available → lock button enabled, green availability, both buttons are enabled.

Figure 18: On the left lot is still available, on the right lot is blocked by Nora BOUKHELIFA which means that she is the only one who can reserve it.

By combining these two sub-tasks we obtain the following rendering

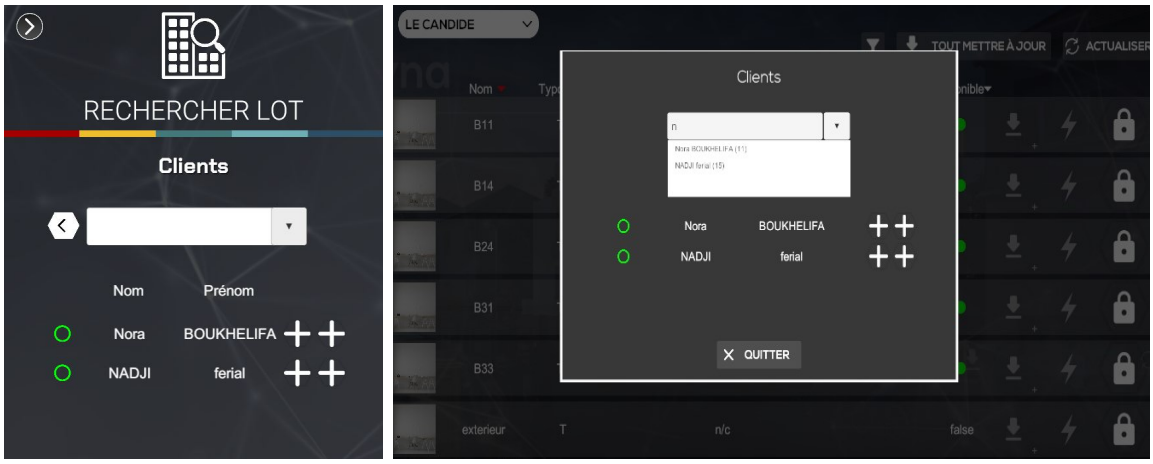


Figure 19: Left lot mode and right program mode (Pop up)

6.4 MapBox

`MapBox`[map, 2019] is the tool used in Nx3D to have a dynamic, interactive and customizable map on the application and to have location data.

The goal of this part is to develop a way to position a house or any construction at the right position.

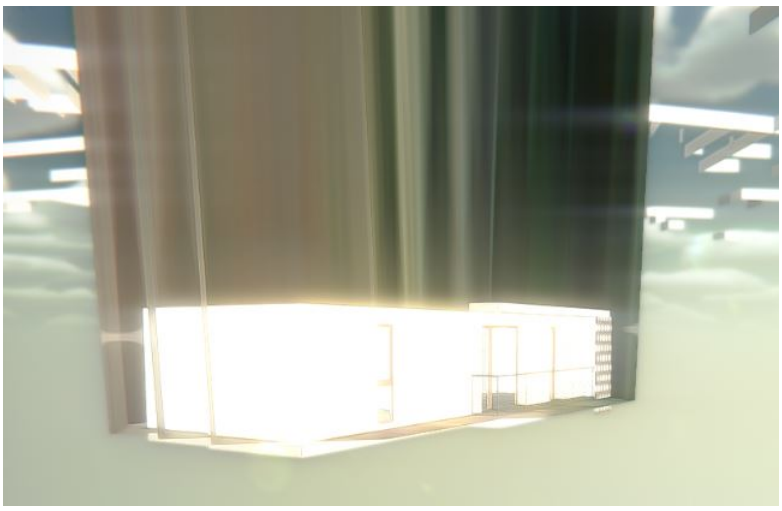


Figure 20: In this figure we can see the rendering before the correction, as we can see that the lot is positioned below the map.

To achieve this goal it was necessary first of all to exploit the MapBox and to find a means of calculating an approximate height between a construction and the map, after reflexion and exploitation of the scripts of mapBox I based myself on the data of the grounds generated by MapBox, to better explain the application Nx3d has already a function of localization which positions the project according to a longitude and a latitude given, when the GPS is activated the MapBox initializes and retrieves the terrain data by interacting with the map API which retrieves the data via google and stores them as children of the different terrains (defined by a certain delimitation) which are in turn children of the MapBox.

Apart from that we wonder what is the link between the generated terrains and the position of the project?

According to the concept of generating MapBox terrains, it orders its child game objects according to the position of the project, from the closest terrain to the farthest, so automatically the project is positioned in the first child (terrain) of the MapBox. So I exploited this principle to calculate the height of the map in relation to the project, making a maximum approximation on the location of the terrain where the project is positioned and its extents, and this by retrieving the maximum point of the mesh boundary, which gives us a 3D vector where the height is only the y coordinate.

Once the height is calculated, we just have to assign the value of the height to the offset and then update the world map to this new offset with the function `UpdateWorldMap(offset)`.

The next step in this task was the `DigGround` function, what does it do? It's a function that takes care of hiding the different objects (slabs, slopes, gardens, etc) for each construction, so that we have a better rendering of the positioning, I want to specify that the function has already been developed for the program visit mode but not for the lots, so I used the same

principle. First of all I developed a `CreateGroundMask()` function if there is no groundMask which creates a new groundMask in the project, and the latter searches for objects of type slab, garden, terrace by duplicating them in a parent named GroundMask. Otherwise the search is done according to the structure of the lot: attic lot, multiplex lot, first floor lot(rdc) or garden lot(rdj).

Note that the digground function also checks if the project is in intersection with other houses which requires a check of each edge of each object of type " habitation " in all the MapBox terrains with each object which composes the project (walls, floors, windows.....) and that to calculate the distance between them, and according to this last one we check if they are in intersection, but considering the number of objects to check and the number of loop to make, the function is very expensive in time what delays to see the result. To overcome this problem and optimize it I used the function `onTriggerEnter()` from unity, it's a predefined function that detects if an object is in collision with another object, it was then enough to add just the action to do if it is in collision. However, I created a script `DestroyHabitation` that I attach to the mesh of the project, in this script I implemented the function `onTriggerEnter()` (code below), if the project collides with an object of type "habitation" it deactivates it.

```
public void OnTriggerEnter(Collider collision)
{
    layer = LayerMask.GetMask("Habitation");

    if (collision.gameObject.layer == 20)
    {
        collision.gameObject.SetActive(false);
    }
}
```

Once we got the result we were satisfied but there was a problem Figure 21, as we calculate the border in relation to a whole land which is not flat which contains elevations, the position of the lot is approximate what makes with the `DigGround()` function we have a kind of rise if position is higher and hollows if it is lower.

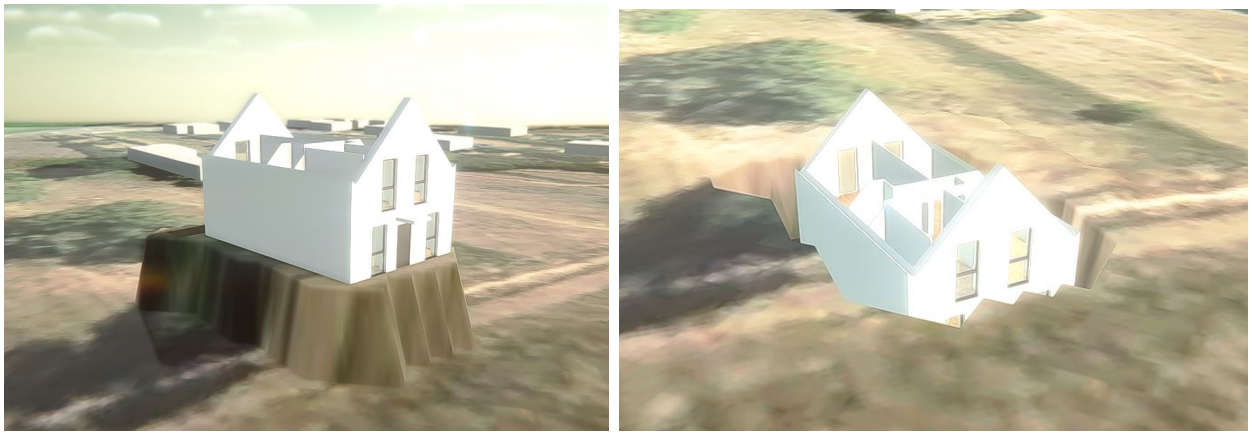


Figure 21: Positioning results, on the left we have a mount and on the right a depression

The solution was to implement in the application a way that allows users to reposition or correct the position of the project, so once we click on the `"GO!"` button the project is positioned and two other buttons are displayed `"reposition"` and a toggle to activate if we want to save the new position.

However, if we want to reposition the project we click on the reposition button and a modification panel is displayed. This panel contains different modes that are managed by `Transform Gizmos`(is a scripting API which allows to easily incorporate transform gizmos at runtime (in-game))[Unity, b]:

- Move mode : When activated, the program places the move handle in the center of the screen. When the user clicks on the arrow handle and starts to move the mouse along the (x,y,z) axis, the terrain is systematically moved to the same coordinates as the mouse. To link the position of the mouse with that of the MapBox offset, the program calculates the position aimed by the pointer on the ground and moves it there.
- Rotation mode: This mode works on the same principle as the move mode, even if the behavior of the associated handle is slightly different. When the user moves the cursor, the interactive ring rotates by the angle formed by the mouse between its current position and its initial position.
- Redo mode: This mode is used to undo the last action performed.
- Undo mode: This mode is used to return to the last action performed.
- Reset mode: This mode is used to reset the initial position of the project.

Figure 22: Localization of the project

To avoid that the area of movement and rotation are limited to the edges of the screen , I made sure that the position of the cursor is linked to that of the project. Once the repositioning is finished, the user just has to validate the modification by clicking on `GO!` and if he wants to save the position before clicking on `GO!` you have to activate the toggle.

How does this save work ? I implemented a `SavePostion()` subsection C.1 function which creates a "position" file which stores the position, rotation and scale of the project in case the user wants to save.

Moreover, I made sure that the `updateOffset()` function that is called when clicking on `GO!` button takes into account the saved position, for this I implemented a `readPosition()` subsection C.2 function that reads the new position if the lot already contains a saved position, so I defined two states for the updateOffset:

- The "UpdateOff" state, if a position has been saved, the data is read and then the batch is positioned directly according to it otherwise the offset will be calculated according to Figure 6.4
- The "UpdateOn" state, represents the state where the user who takes care of the repositioning.

6.5 Configuration window

The purpose of this task is to develop functions to control the parameters of the application such as sound, mouse and camera.

Sound configuration

For the sound parameters the goal is to classify the output audio according to a category:

- Master which represents all the audio
- Ambiance designates all the audio of the environment (rain, wind, birds singing...etc)
- Menu is the audio of the different buttons (click or hover) and the audio of the animations.

Figure 23: Sound configuration

Once the sound is mixed into these categories, effects and other operations can be applied to these categories as a whole. This is powerful not only in applying game logic changes to the different categories of sound, but also in allowing designers to modify different aspects of the mix to achieve what is called "mastering" of the entire soundscape at runtime.

To realize this I based on the concept of the AudioMixer which is an asset that can be referenced by AudioSources(is a component that allows to play sounds in the scene) to provide more complex routing and mixing of the audio signal generated by AudioSources[Unity, 2020].

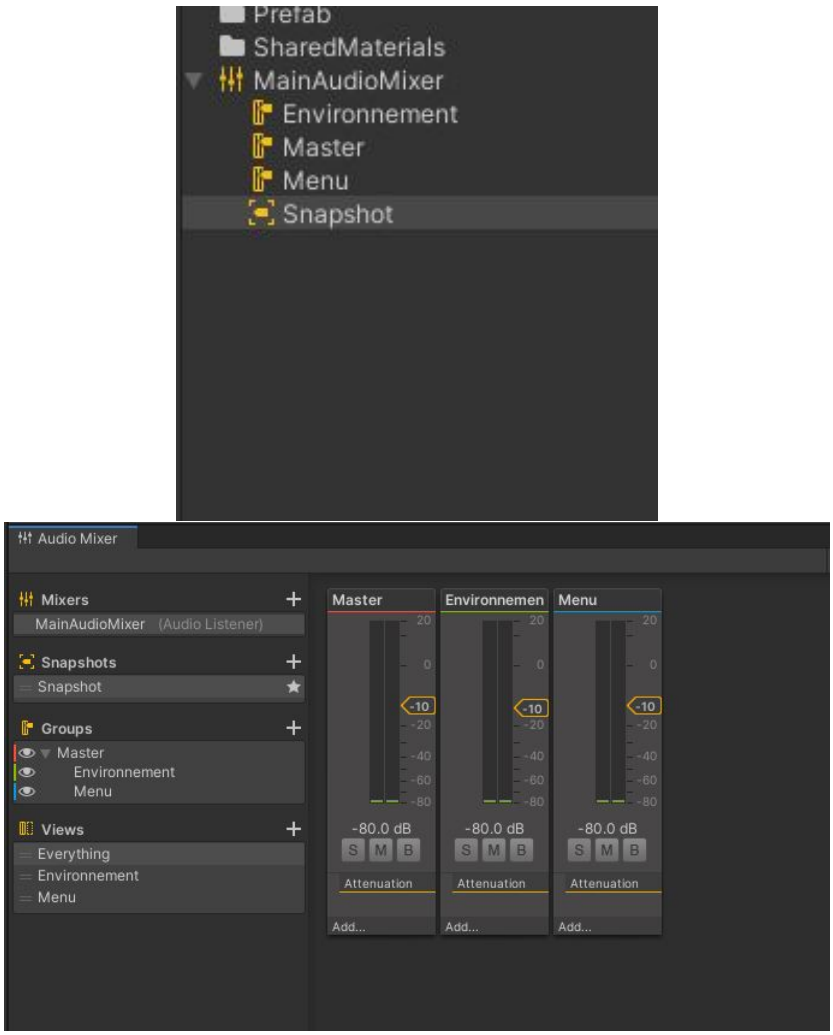


Figure 24: Audio Mixer

As can be seen from Figure 24 I specified two groups: Menu for menu audios and Environment for ambient audios.

Each game object that is supposed to produce a sound in the application has an AudioSource component and depending on the type(Menu/Environment) of sound, an AudioListener(a component that, as the name suggests, listens to the sound in your scene (that is played from an AudioSource) is assigned to it.

However, using the snapshots that allow to capture the state of an AudioMixer and the transition between these different states as the application progresses, we retrieve the volume parameter and then I assign the value defined by the user in the application, and we apply this for the master, menu and ambience, in the case where we want to mutate the sound it was enough to put the volume to 0, the code below summarizes how the volume of the Master was controlled and we did the same for the other categories .

```
//value defined by the user
var volumeMaster = SliderMasterVolume.SliderValue;
```

```
AudioListener.volume = volumeMaster;
//assign value to mixer
mixer.SetFloat("MasterVolume", LinearToDecibel(volumeMaster));
```

`LinearToDecibel()` is a function I implemented to convert the retrieved value to decibel like the volume in Audiomixer is in decibel.

```
private float LinearToDecibel(float linear)
{
    float dB;
    if (linear != 0)  dB = 20.0f * Mathf.Log10(linear);
    else  dB = -144.0f;
    return dB;
}
```

Mouse and camera configuration

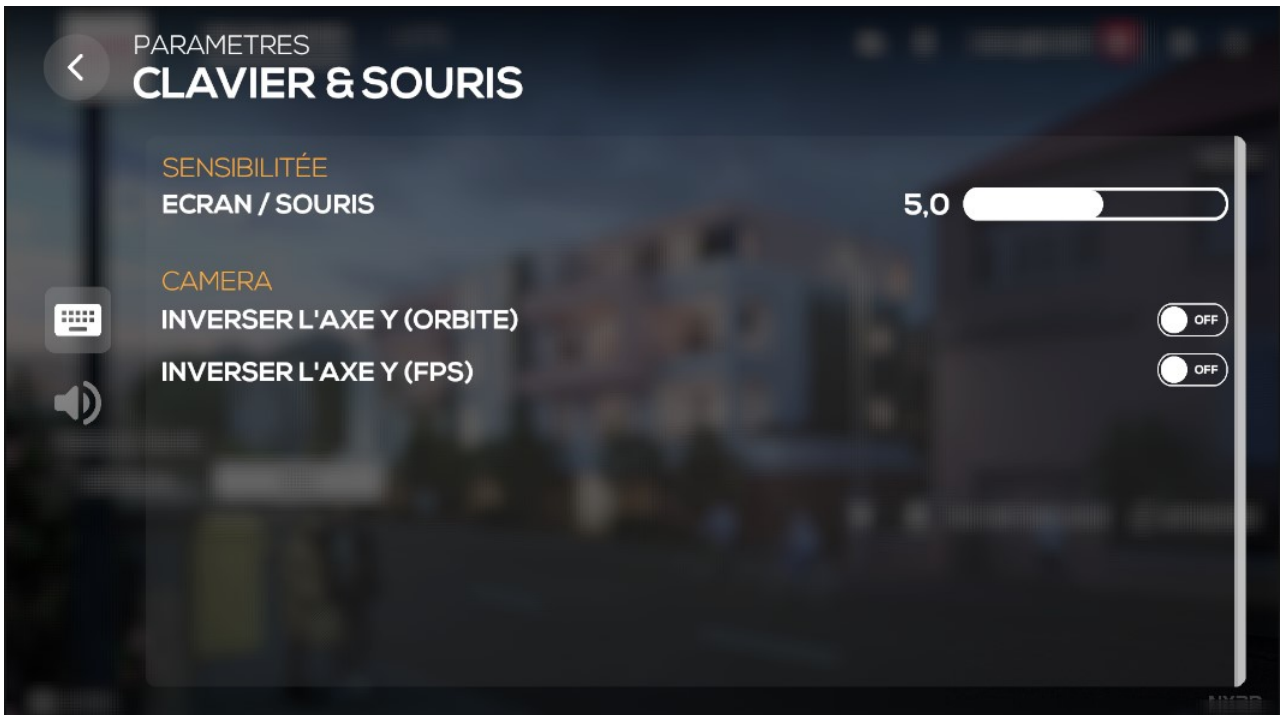


Figure 25: Mouse configuration

In the mouse configuration, we want to give users the control to adjust the mouse speed to make it comfortable because the sensitivity of the movements is sometimes quite tricky to manage. Indeed, the difference in movement amplitude between a movement with the sticks or with the tilt of the controllers is quite important. It is therefore necessary to adjust the sensitivity before launching the task or during the game.

To do this, I get the camera that renders according to the eye of the screen and then I get the `SpeedAxis` variable and I assign it the value set by the user that I get from the slide value.

```
public void ChangeMovementSpeed()
{
    //value defined by the user
    float speedMovement = SliderMovement.value;
    CameraMovement.SpeedAxis = new Vector2(speedMovement*10,
      speedMovement*10);
}
```

Regarding the camera configuration the goal is to allow them to reverse the y-axis because for some people the reversed controls make much more sense, they press the top to levitate the camera higher, which will direct the perspective downwards while keeping the subject in the center of the frame.

And as we have two view modes: FPS mode for the immersive view and the orbital mode so we have the possibility to reverse either one or both, however I implemented two functions Appendix E to reverse the y axis and the principle is to check in which mode we are (FPS/orbital) then use the speed axis variable, and depending on the mode we multiply the y coordinate of the speed vector by -1.

6.6 Other adjustments

Weather and Sunset

Nx3d has chosen `Enviro - Sky and Weather` [env, 2020] as its weather system, it is a unity asset that gives the possibility to create our own weather types and to control light, sky, fog and clouds.

For the management of weather and environment, Nx3D has opted for a change of weather according to a calendar but the concern that arose that no matter the month we choose the weather and sunshine are not updated, in order to overcome this problem I documented the asset `Enviro - Sky and Weather` to understand its operation, I managed to develop a function based on what I exploited as code of the asset taking into account some variables: number of days in the year to define the 4 seasons, winter/summer time difference (utc) and latitude and longitude of the project.

Figure 26: Meteo configuration

First of all I defined the interval according to day of the year for each season :

- Spring → start:80, end:172
- Summer → start:173, end:265
- Autumn → start:266, end:355
- Winter → start:356, end:79

Then I developed an `UpdateEnviro()` function according to a date as you can see below

```
public void UpdateEnviro(DateTime date)
{
    GetDayOfYear(date);
    if (DayNumber >= 87 && DayNumber <= 304)
        enviro.GameTime.utcOffset = 2 ;
    else
        enviro.GameTime.utcOffset = 1;

    if (SelectedLot!=null && SelectedLot.gps!=null)
    {//gps for lot visit
        LatLonUser = SelectedLot.gps;
```

```
    }else if (SelectedProgram!=null && SelectedProgram.gps!=null
      )
    {//gps for program visit
        LatLonUser = SelectedProgram.gps;
    }else
    {//custom gps values
        LatLonUser = m_gps._x.text+ "," + cm_gps._y.text;
    }

    enviro.GameTime.Latitude = (float)LatLonUser.x;
    enviro.GameTime.Longitude = (float)LatLonUser.y;
}
```

`UpdateEnviro()` controls the environment and the weather according to the date, it contains the function `GetDayOfYear()` which recovers the number day of the year, then according to this days we define the utcOffset which allows the edition of UTC timeoffset, then according to the launched project (program or lot) we recover its latitude and longitude then we assign them to GameTime.Latitude and GameTime.Longitude respectively to represent the position of the sun.

```
public void GetDayOfYear(DateTime date){
    DateTime dateToDisplay = date;
    DayNumber = dateToDisplay.DayOfYear;
    EnviroSkyMgr.instance.SetDays(DayNumber);
}
```

This process worked perfectly except while doing tests I noticed that there is a problem on the calendar any day selected on the calendar it refers to the same day, however I modified the function so that when we click on a day the date is retrieved knowing that the days are only items on unity, and once we retrieve the date in text form we convert it to Date format then we call the `UpdateEnviro()` function for an update.

```
public void OnDateItemClick(string day){
    DateTime date = new DateTime(Int16.Parse(_yearNumText.text),
       Int16.Parse(_monthNumText.text), Int16.Parse(day));
    UpdateEnviro(date)
    GetDayOfYear(date);
}
```

Measure objects

This task consists of measuring all types of objects contained in a construction such as wall, floor, ceiling ..., for that it was enough to attach the script named "`measure`" on each object.

`measure` is a script that contains all the functions to calculate the dimension of an object: height, width and depth and this by recovering the boundary of the mesh or collider with the unit of choice cm, m, km

```
width_meters = bounds.size.x;
height_meters = bounds.size.y;
depth_meters = bounds.size.z;

width = unitConversion(width_meters);
height = unitConversion(height_meters);
depth = unitConversion(depth_meters);
```

In addition, each object has a Behaviour component that allows it to be activated or deactivated, which gives the object the possibility of being detected and duplicated once the user fixes the cursor on it to modify it or see its information [Unity, 2020].Note that the duplication is temporary, i.e. it is destroyed when the cursor is on another one.

Moreover to better identify the object I added a function Appendix D which recovers the name of the object, one distinguishes two types:

- The object is an element of the supplies i.e. of the library, I made sure to recover its ID which seeks it in list of the libraries.
- The object is an element of the architecture of the project (wall, floor, ceiling ...), in this case I get the name of gameobject.

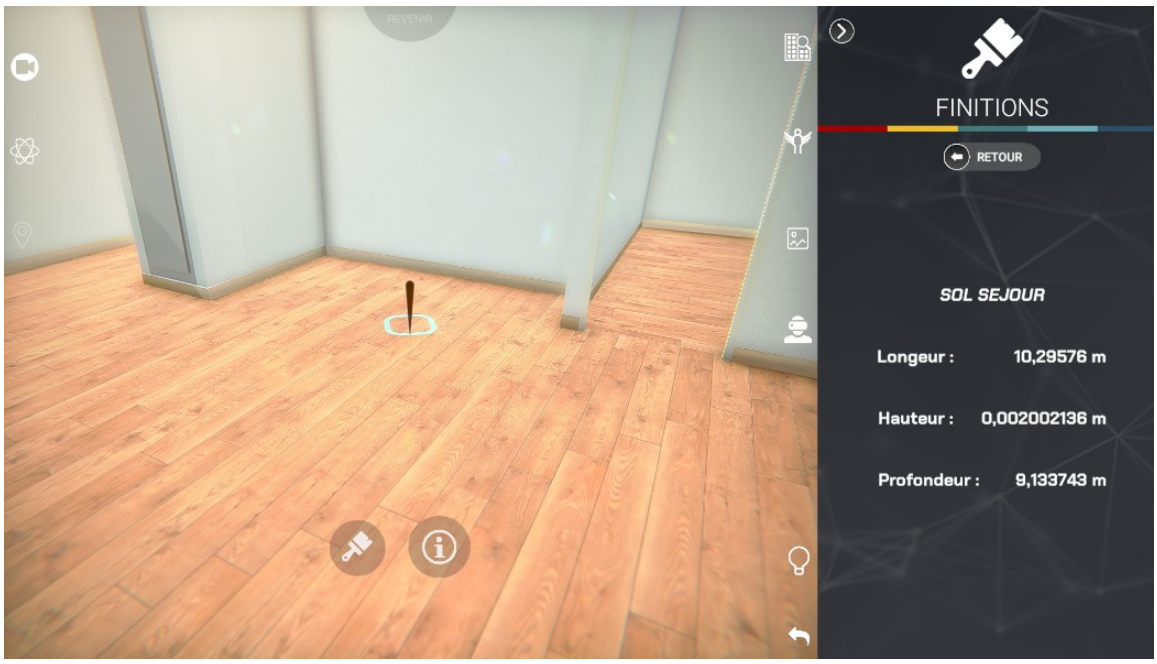


Figure 27: This figure shows us an example, we can see the cursor which is fixed on the ground and in the right part of the image we can see the information of the ground.

Filter the lot

This task consists in exploring what has already been done in the filtering of lots according to different categories of filters: price, surface and number of rooms and additional filters that are floor, orientation, presence of garden and terrace, these categories of filters that our search provides cover the most important aspects of what users will filter, according to expanded needs.

Moreover, each filter in the additional filters is added by the "add filter" button and deleted by `"X"` button, so the user has to click each time on button to add a category which does not simplify the application, hence the instruction of adding all filters with a single click. Besides the additional filters on Unity are a prefab named `AdditionnalFilter` with the categories that are children, and for each addition of a filter we duplicate this prefab and according to the category "x" we activate the child gameOobject "x" and we disable the rest.

To correct this I had to exploit the code to modify it, and to generate the additional filter of a single click by making a loop on the list of the filters, and modify all the functions of adding, deleting, refreshing ..., thereafter I fixed constraint with regard to operation

- Firstly, the added filters are stored in a new list `additionnalFilterlist`, if this one contains all the filters then the button `add filter` is not selectable anymore.
- Secondly, we have another list `removedFilterList` which stores the filters removed by the user (click on "X"), if the number of elements in this list is >0, then `add filter` becomes selectable and by clicking we add the filters not displayed.

- Thirdly, for the orientation and floor categories of the filters I added an empty field in their choice of values, such that the empty field implies that the category is not taken into account and for this I implemented a few extra lines to handle this.

- Finally, I made the validation button activate another button which is the reset button, and this button is used to reset the filters and go back.

Moreover when I had to implement and modify this functionality, I decided to improve the display and animation of this last one by adding a button to reduce the additional filters.

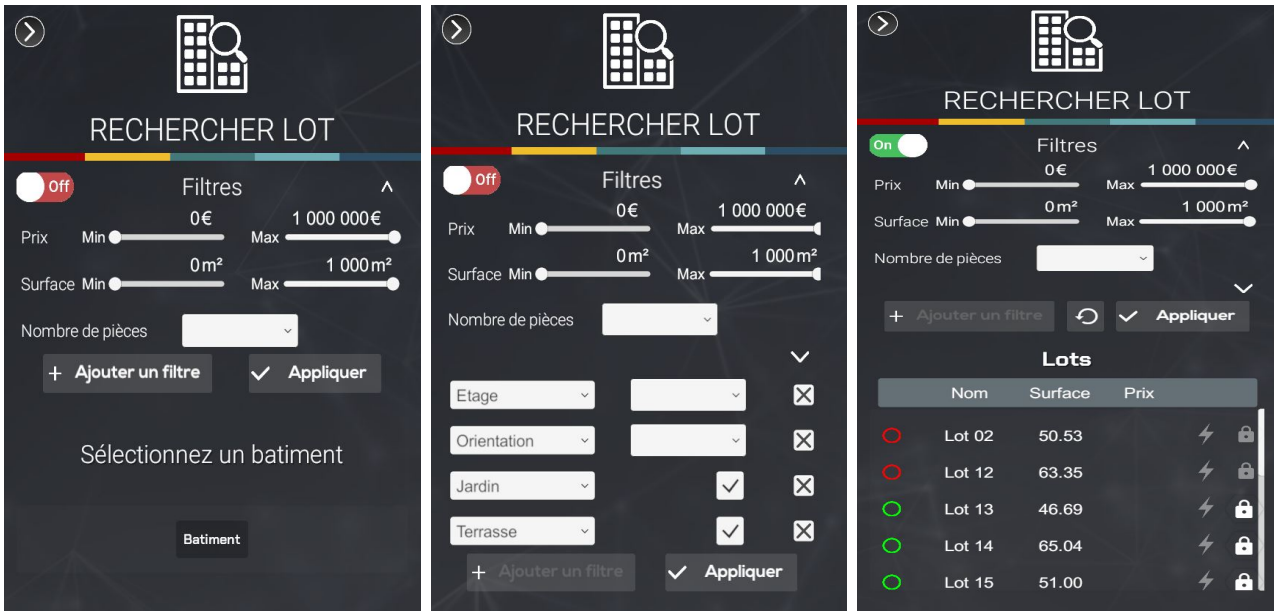


Figure 28: An overview of the lot filter window.

Graphic redesign of the interface

In addition to the various features I designed to meet the objectives, I also reorganized the newly added elements and the elements already present in the interface to make them more discrete. I took advantage of this task to design a structure in which existing or future features could be integrated.

To readapt the new features to what is already present in the interface of the application I added in the case of reservation of the lot has Lock confirmation Pop up that appears when you click on the button option or reservation of the lot. I also made sure to put animations of opening and closing during the display of this pop up and the window of customer search bar for the lock of the lots as well as their closures .

Figure 29

In addition, to allow users to better see if the lot is locked or not I made sure that the availability of the lot takes a color according to its status: green (available), orange (lot has an option) and red (reserved) Figure 30.

Figure 30: The addition of the color for the availability of the lots

Moreover I implemented a function Appendix F which allows to sort the lots by ascending/decreasing order according to the availability such that the order is from available to unavailable

Figure 31: The lots ordered in ascending order according to their availability.

In the same way for the pastilles put on the lots I manage the color of shader of the pastille according to the state of the lot Figure 32.

Figure 32: The figures above show us the pastilles in a certain color depending on the availability of the lot in which it is placed, such as the figure on the left represents a view on the building and right a view on a floor.

7 Conclusion

This internship at NX3D brought me a lot of skills while contributing to the development of a long term project.

Indeed, I was able to learn how to use a 3D video game engine and a new programming language. Although this environment is not directly taught in my studies, it allowed me to learn about the concept and logic of such a development platform.

But the skills brought by such a project are not only technical, they are also relational. I learned that the sharing of information was more important than ever, especially in the case of interconnected projects.

Both on the technical and the human level, the methods and working methods I have acquired will be essential in my future personal or professional projects. Indeed, I was able to consolidate my experience on "research" oriented work while discovering a tool widely used by companies and laboratories.

Moreover, the most interesting thing for me is the virtual reality which is a fast growing field and whose future applications will be really incredible.

In conclusion, this internship has been enriching on all levels, both technical and human, I am quite satisfied with the work done because the set objectives have generally been achieved.

References

[map, 2019] (2019). *Maps for Unity*.

[env, 2020] (2020). *Enviro - Sky and Weather*.

[Andre, 2017] Andre, G. (2017). Rapid adaptation to a non-individualized binaural spatialization : case of virtual reality virtual reality applied to video games.

[B., 2020] B., G. (2020). Five innovative ways you can use virtual reality in the real estate business.

[Contributors, 2021] Contributors, M. (2021). An overview of http.

[G., 1710] G., B. (1710). Principles of human knowledge.

[Gang,] Gang, W. The study and development of 3d virtual simulation system for the gas pumping and condensing sections of coking plant.

[Informatique,] Informatique, M. *ArchiCAD*.

[Nandaa,] Nandaa, A. A look at http request methods.

[NOVAE, 2019] NOVAE (2019). *Unity 3D, the essential tool for multimedia creation*.

[NX3D, 2019] NX3D (2019). Scotty platform.

[Platon, vJ C] Platon (IV siècle av.J.-C). *ALLEGORY OF THE CAVE*.

[R., 1637] R., D. (1637). Discourse on the method.

[Unity, a] Unity. Coding in c# in unity for beginners.

[Unity, b] Unity. *Positioning GameObjects*.

[Unity, c] Unity. *UnityWebRequest*.

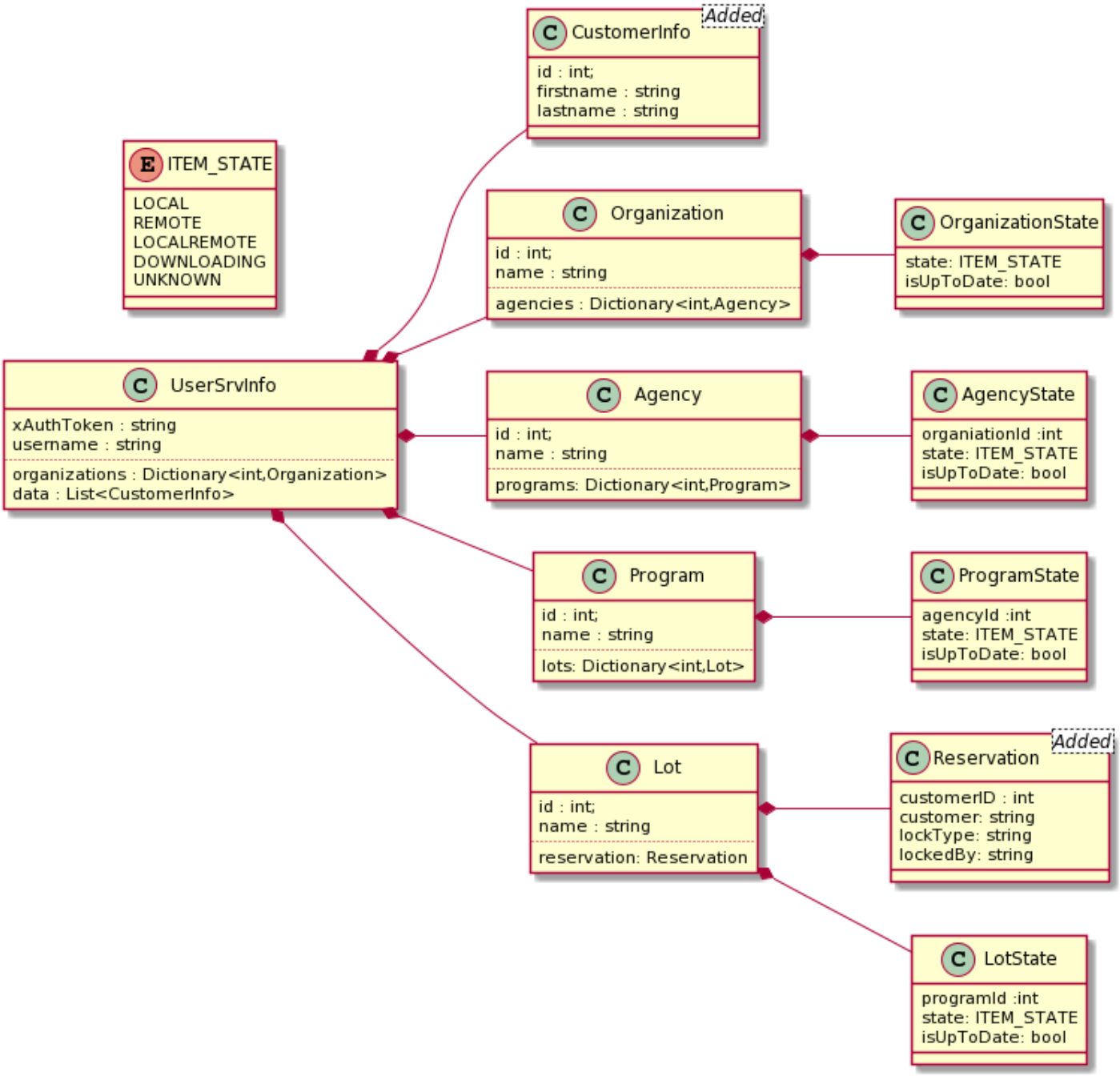
[Unity, 2020] Unity (2020). Unity user manual 2020.3 (lts).

[UQO,] UQO. Laboratoire de cyberpsychologie de l'uqo,brief history of virtual reality.

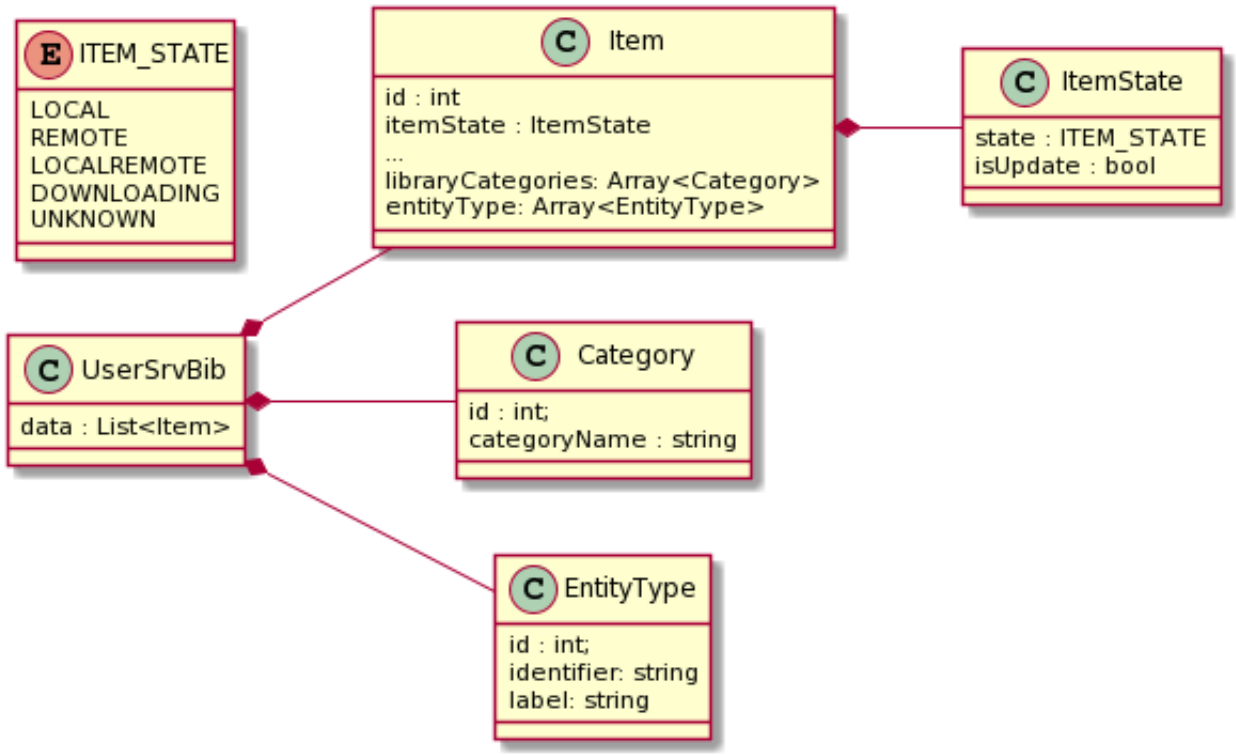
[Yaodong Sun, 2018] Yaodong Sun, Xi Chen, Z. H. D. T. L. C. C. Z. L. L. Q. Z. (2018). Application of virtual reality technique to civil engineering. *Atlantis Press*.

[Zhonghua, 2004] Zhonghua, J. X. L. (2004). Present situation of vr researching at home and abroad.

A User data architecture



B Library data architecture



C MapBox

C.1 Save Position

```
public void CreatePosition(Lot lot ,GameObject MapBox )
{
    GameObject Obj = MapBox.transform.parent.gameObject;

    string P = Obj.transform.position.x.ToString() + "." + Obj.
      transform.position.y.ToString() + "." + Obj.transform.
      position.z.ToString();
    string R = Obj.transform.localEulerAngles.x.ToString() + "."
       + Obj.transform.localEulerAngles.y.ToString() + "." + Obj
      .transform.localEulerAngles.z.ToString();
    string S = Obj.transform.localScale.x.ToString() + "." + Obj
      .transform.localScale.y.ToString() + "." + Obj.transform.
      localScale.z.ToString();

    NewPosition pos = new NewPosition() {
        position = P,
        rotation = R,
```

```
        scale = S
    };

    NewPosition data=SrvData.UserSrvInfo.DeepCopy<NewPosition>(
      pos);
    string json=Newtonsoft.Json.JsonConvert.SerializeObject(data
      );
    File.WriteAllText(Path.Combine(lot.localPath,"position"),
      json);
}
```

C.2 Read Position

```
public static NewPosition ReadOffset(string filePath)
{
    if (File.Exists(filePath))
    {
        Debug.Log("[SetGPS::ReadPositionFromFile] - Attempt to
          read datas from " + filePath);

        string jsonDataContent = File.ReadAllText(filePath);
        if (jsonDataContent != null && jsonDataContent.Length >
          0)
        {
            NewPosition datas = (NewPosition)Newtonsoft.Json.
              JsonConvert.DeserializeObject(jsonDataContent,
              typeof(NewPosition));

            if (datas != null)
            {
                return datas;
            }
        }
        return null;
    }

    return null;
}
```

D Measure Objects

```
private string GetObjName(GameObject obj){
    string name = null;
    foreach (string type in ObjectType)
    {
        if (obj.name.StartsWith(type))
        {
            name = type.Replace("_"," ").ToUpper();
        }
    }
    if (name == null)
    {
         if (obj.GetComponentInParent<PartBehaviour>() != null)
         {
            UpdaterLibrary updaterLibrary = m_updateCenter.
              GetUpdaterLibrary();
            name = GetObjLabel(updaterLibrary.Items,obj).ToUpper
              ();
         }
    }
    return name;
}

public string GetObjLabel(List<UserSrvBib.Item> items ,
  GameObject obj){
    string nameObj = null;
    if (items != null)
    {
        foreach (UserSrvBib.Item item in items)
        {
            if (item.enabled)
            {
                if (item.entityType != null && item.entityType.
                  Length > 0)
                {
                    int itemId = item.replacement_id;
                    string name = item.label;
                    if (name == null || name.Length == 0)
                    {
```

```
                        name = item.filename;
                    }
                    if (obj.GetComponentInParent<PartBehaviour
                      >() != null)
                    {
                        if (obj.GetComponentInParent<
                          PartBehaviour>().Id == itemId)
                        {
                            nameObj = name;
                        }
                    }
                }
                else
                {
                    Debug.LogWarning("[CreationMenuManager::
                      InitializeItems] - Item " + item.id + "
                      has no entityType");
                }
            }
        }
    }
    return nameObj;
}
```

E Camera Configuration

```
public void InvertYAxisOrbit(bool FlipY){
    if ((IsCameraOrbital) && (CameraOrbit.SpeedAxis.y > 0))
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == true ? -1 : 1);
    }
    if ((IsCameraOrbital) && (CameraOrbit.SpeedAxis.y < 0))
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == false ? -1 : 1);
    }
    if ((IsCameraFPS) && (FPSOn == false) && (CameraOrbit.
      SpeedAxis.y < 0))
```

```
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == false ? -1 : 1);
    }
}
public void InvertYAxisFPS(bool FlipY){
    if ((IsCameraFPS) && (CameraOrbit.SpeedAxis.y > 0))
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == true ? -1 : 1);
    }
    if ((IsCameraFPS) && (CameraOrbit.SpeedAxis.y < 0))
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == false ? -1 : 1);
    }
    if ((IsCameraOrbital) && (OrbitOn == false) && (CameraOrbit.
      SpeedAxis.y < 0))
    {
        CameraOrbit.SpeedAxis.y = CameraOrbit.SpeedAxis.y * (
          FlipY == false ? -1 : 1);
    }
}
public void SetOrbitalCameraMode ( ) {
    FPSOn = SwitchFPS.transform.GetComponent<SwitchManager>().
      isOn;
    OrbitOn = SwitchOrbit.transform.GetComponent<SwitchManager
      >().isOn;
    SetOrbitalCameraMode ( OrbitMaxDistance );

    InvertYAxisOrbit(OrbitOn);
    InvertYAxisFPS(OrbitOn);
}

public void SetFPSCameraMode ( ) {
    FPSOn = SwitchFPS.transform.GetComponent<SwitchManager>().
      isOn;
    OrbitOn = SwitchOrbit.transform.GetComponent<SwitchManager
      >().isOn;
    SetFPSCameraMode ( FPSMaxDistance );
```

```
    InvertYAxisOrbit(FPSOn);
    InvertYAxisFPS(FPSOn);
}
```

F Sort lots by the availability

```
case LotSortType.AVAILABILITY:
    relativeValue = PreventEmptyString(t1.Item1.dispo, t2.Item1.
      dispo, () => {
        bool isT1Available = ( t1.Item1.dispo == "true") && (t1.
          Item1.reservation == null);
        bool isT2Available = (t2.Item1.dispo == "true" ) && (t2.
          Item1.reservation == null);

        int relative = 0;

        if (isT1Available && !isT2Available)
        {
            relative = -1;
        }
        else if (!isT1Available && isT2Available)
        {
            relative = 1;
        }
        else
        {
            bool isT1Optioned = (t1.Item1.reservation != null)
              && (t1.Item1.reservation.lockType == "option");
            bool isT2Optioned = (t2.Item1.reservation != null)
              && (t2.Item1.reservation.lockType == "option");

            if (isT1Optioned && !isT2Optioned)
            {
                relative = -1;
            }
            else if (!isT1Optioned && isT2Optioned)
            {
                relative = 1;
            }
            else
```

```
            {
                bool isT1Locked = (t1.Item1.reservation != null)
                   && (t1.Item1.reservation.lockType == "
                  reservation");
                bool isT2Locked = (t2.Item1.reservation != null)
                   && (t2.Item1.reservation.lockType == "
                  reservation");
                if (isT1Locked && !isT2Locked)
                {
                    relative = -1;
                }
                else if (!isT1Locked && isT2Locked)
                {
                    relative = 1;
                }
            }
        }

        return relative;
    });
    break;
```