Aalto University
School of Engineering

# DOCUMENTATION

Project 3D models and window view

v 1.0

GIS-E4030 GIS Development
Chau Ngoc
Jussila Anssi
Snickars Caj
Suominen Eemeli
Aalto University School of Engineering
Master's Programme in Geoinformatics

Spring 2021

# Contents

# Introduction

The aim of this project was to create a simulated view from a building window in Helsinki, which would simulate how the view would in reality look like. The simulated view is created for given street address, floor level and direction. The simulated view does not give the exact location of a window in apartment but more of a generic view from the building.

A simulated view from a building has many applications. It can be used to see the surrounding environment before buying or renting apartments. This type of simulation could also see usage when constructing new buildings. One could see what kind of scenery would be visible from the building or see how the structure would affect the view of nearby buildings. The view from the window could also be used in different analysis. It is possible to calculate the farthest distance from the view or one could calculate the amount of sunlight that reaches the window at different times of day or season. These are just couple applications where this kind of simulation would be beneficial.

This project uses Unity 3D game engine as environment, where the data is visualized, and effects of activated scripts happen. The scripts are written in C# programming language and the version control of the project was done via GitHub. GitHub desktop was utilized to communicate with GitHub from a desktop application.

This project uses a 3D city model, address registry of Greater Helsinki region and an elevation model as data. All of the data are public, and the provider is city of Helsinki.

# Quickstart guide

This is a short guide which presents how to the program works.

## Features

The menu used in operating the software locates in the top-left corner. The features of the menu are listed in the table below and from the figure 1 you can see the actual menu.

| | | | |
|---|---|---|---|
| ❶ | "Enter an address…" – input text field | ❺ | "Forward"-button |
| ❷ | "Enter a floor…" – input text field | ❻ | "Backward"-button |
| ❸ | "Enter a direction…" – input text field | ❼ | "Distance"-button |
| ❹ | "OK"-button | | |


*Figure 1. Screenshot of the software displaying the menu*

## Search for an address

To get a window view of an address, you need to fill the three input text fields and click "OK"-button.

❶ **"Enter an address…" – input text field:** Write the exact address to the field including the street number if the address has one.

❷ **"Enter a floor…" – input text field:** Enter the wanted building floor (e.g., for the 1st floor type "1").

❸ **"Enter a direction…" – input text field:** Enter the wanted direction in degrees (Note that: North=360°, East=90°, South=180°, West=270°).

❹ **“OK”-button:** After filling the three text fields and clicking “OK”-button, wait for couple of seconds and the camera will be directed in the wanted location, direction and building floor for the window view as can be seen in the example (figure 2).

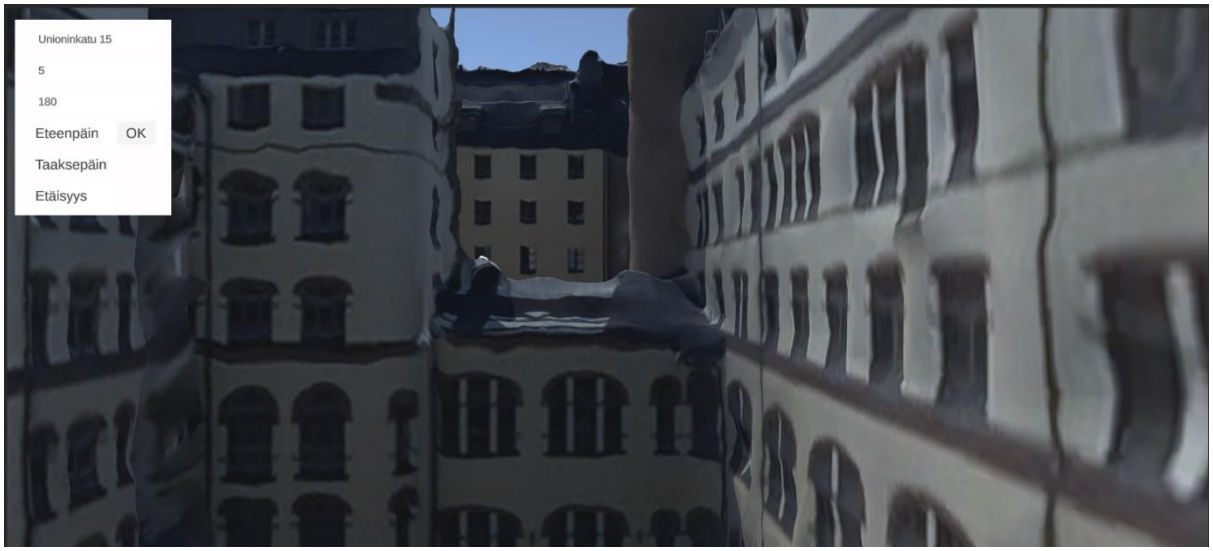



*Figure 2. The south window view from Unioninkatu 15 and 5th floor.*

## Shifting between different buildings

After setting the wanted address, building floor, and viewing direction, it is possible to move the camera to the following or previous buildings. This happens with ❺”Forward”-button and ❻ ”Backward”-button. From the figure 3 and 4, can be seen how the position of the camera and window view changes after clicking the ❺”Forward”-button. Respectively, the position goes right back to the initial position when clicking ❻”Backward”-button. These buttons can be used several times, so it will direct to the next building.

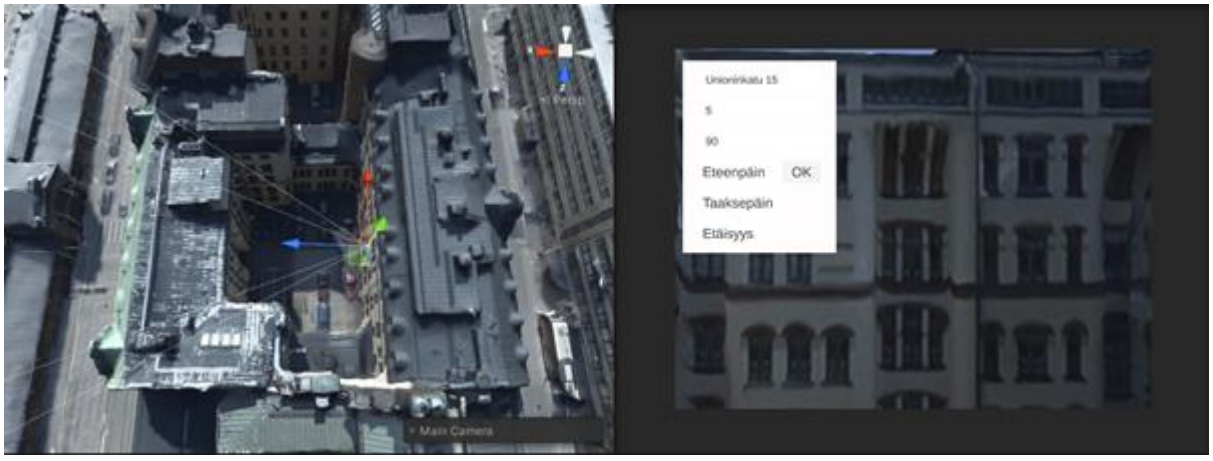



*Figure 3. The initial position of the camera (left) and the initial window view (right).*

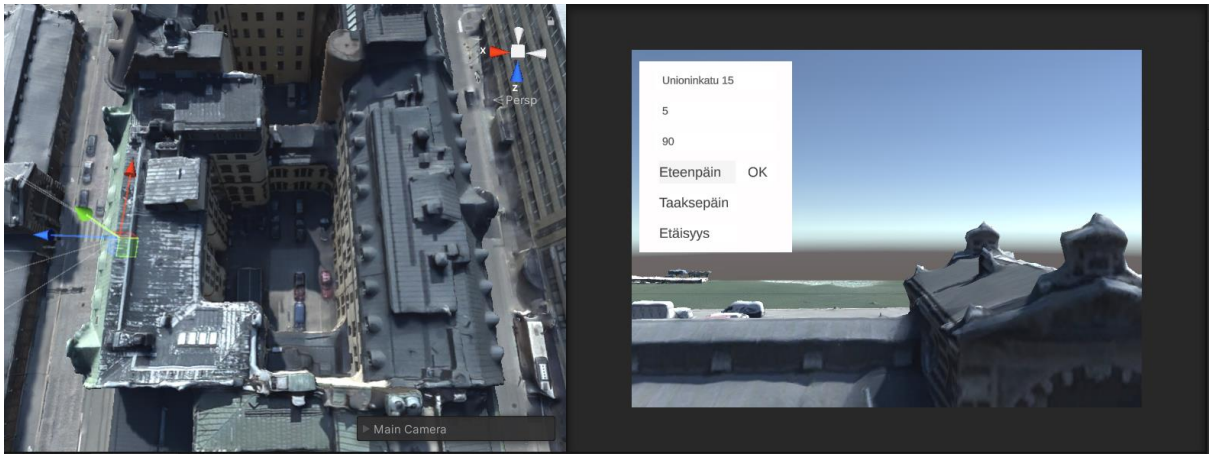



*Figure 4. The new camera position and window view after clicking the "Forward"-button.*

## Measuring the furthest distance from the window view

In addition to simulating the window view, it is possible to measure the furthest distance visible in the view by clicking the ❼ "Distance"-button. A red pin will appear, and it locates the furthest point. The distance is shown above the pin in meters (figure 5).

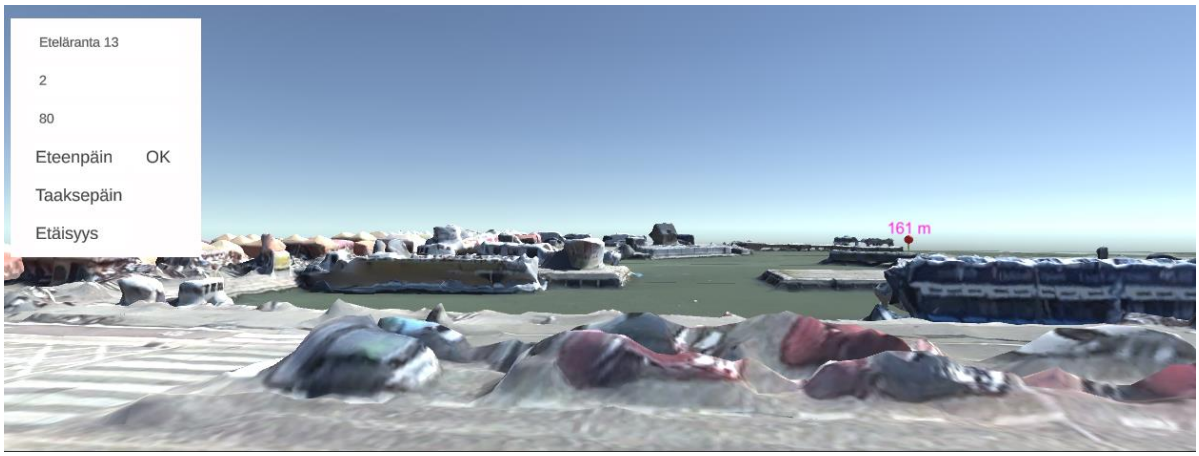



*Figure 5. "Distance"-feature in action displaying the furthest distance visible in the view.*

## Moving the view by WASD keys and mouse

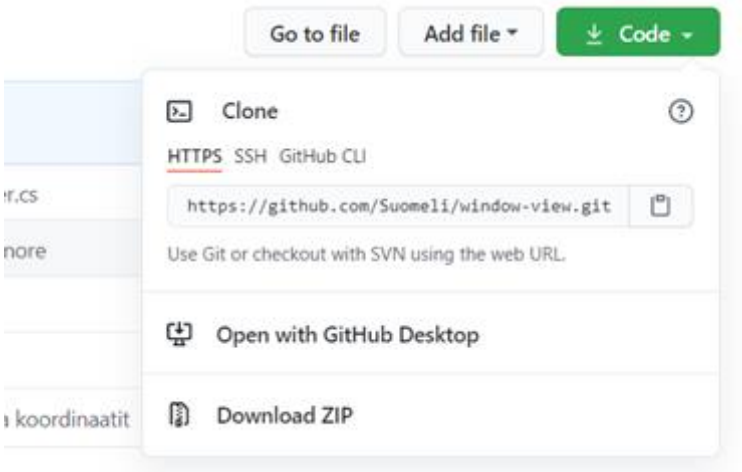
There is also a mode for free movement of the camera which is called Free Mode. To activate the Free Mode, press "F"-key from your physical keyboard. In Free Mode, you can change the viewing angle by simply moving the mouse cursor in the wanted direction and with WASD-keys you can move in horizontal directions. You can exit from the Free Mode by pressing the "F"-key again.

## Get access to the GitHub Project with Unity

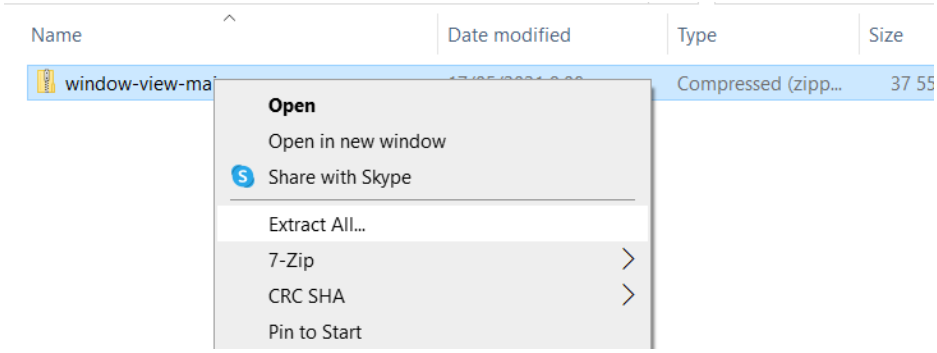
This section will guide you on how to get the source code of the project in two different ways. The first method is fetching the project as a zip file without needing to install Git or GitHub Desktop client. The second method is to access the project by cloning the git repository to your local machine with GitHub Desktop. After fetching the project to your computer, this guide will show how to open the project in Unity.

### Get the source code as a zip file

1. Go the main page of the repository with this link: https://github.com/Suomeli/window-view
2. Click the “Code”-button which locates above the list of files
3. Click “Download ZIP”



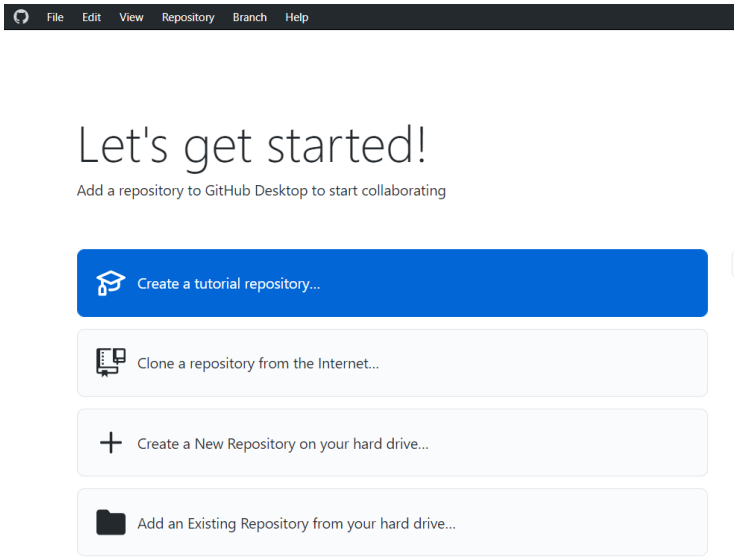

4. After downloading the zip file, right-click it to select “Extract All..” and a window pops up where you can enter the path you wish to extract the files to.



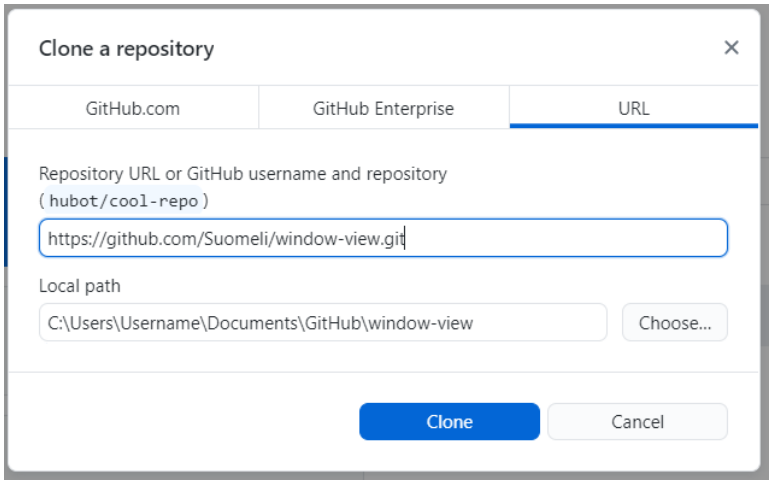

5. Now you have the source code of the project in the folder path you chose.

## Get the source code by cloning the repository

1. Download the GitHub Desktop client in here: https://desktop.github.com/
2. Go the main page of the repository with this link: https://github.com/Suomeli/window-view
3. Click the “Code”-button which locates above the list of files
4. Copy the HTTP link
5. Open the Desktop GitHub and click “Clone a repository from the Internet…”



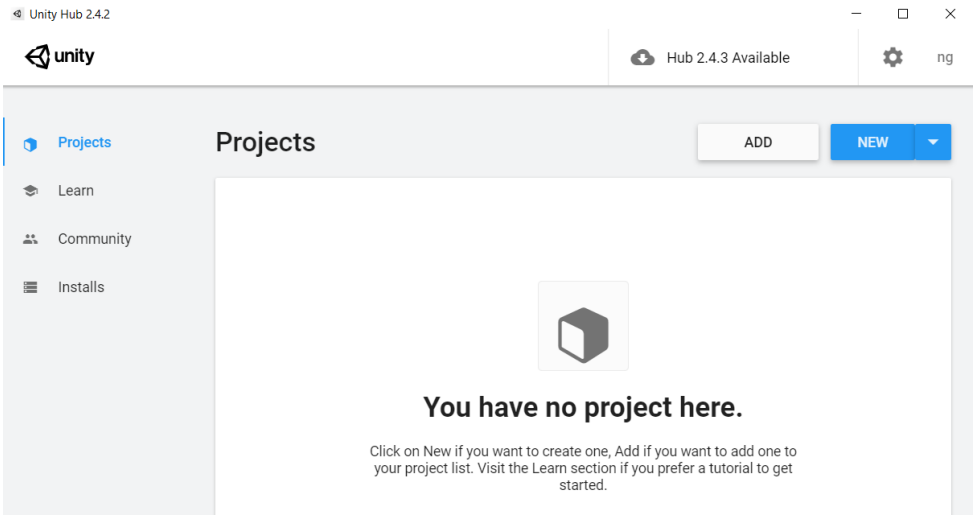

6. Click the “URL”-tab and paste the HTTP link from the GitHub main page and choose the local path as you wish. After this, click “Clone”.



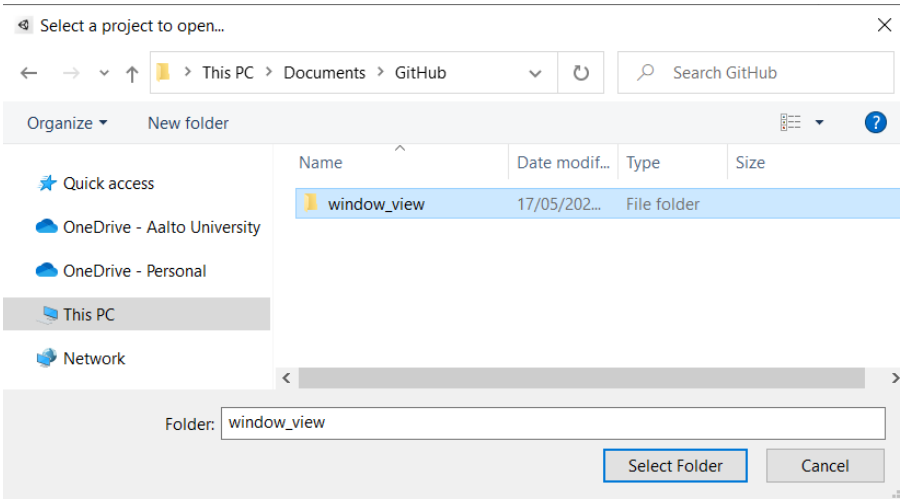

7. Now you have the source code of the project in the folder path you chose.

## Open the project in Unity

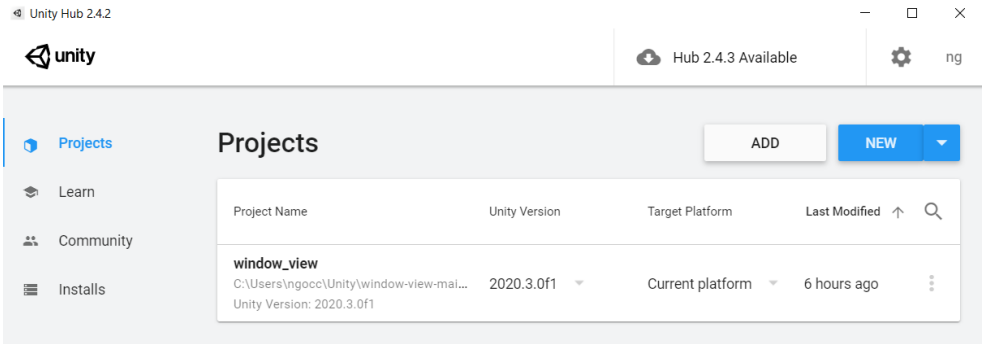
1. Download Unity Hub from here: https://unity3d.com/get-unity/download
2. Open the Unity Hub and then click “ADD”.



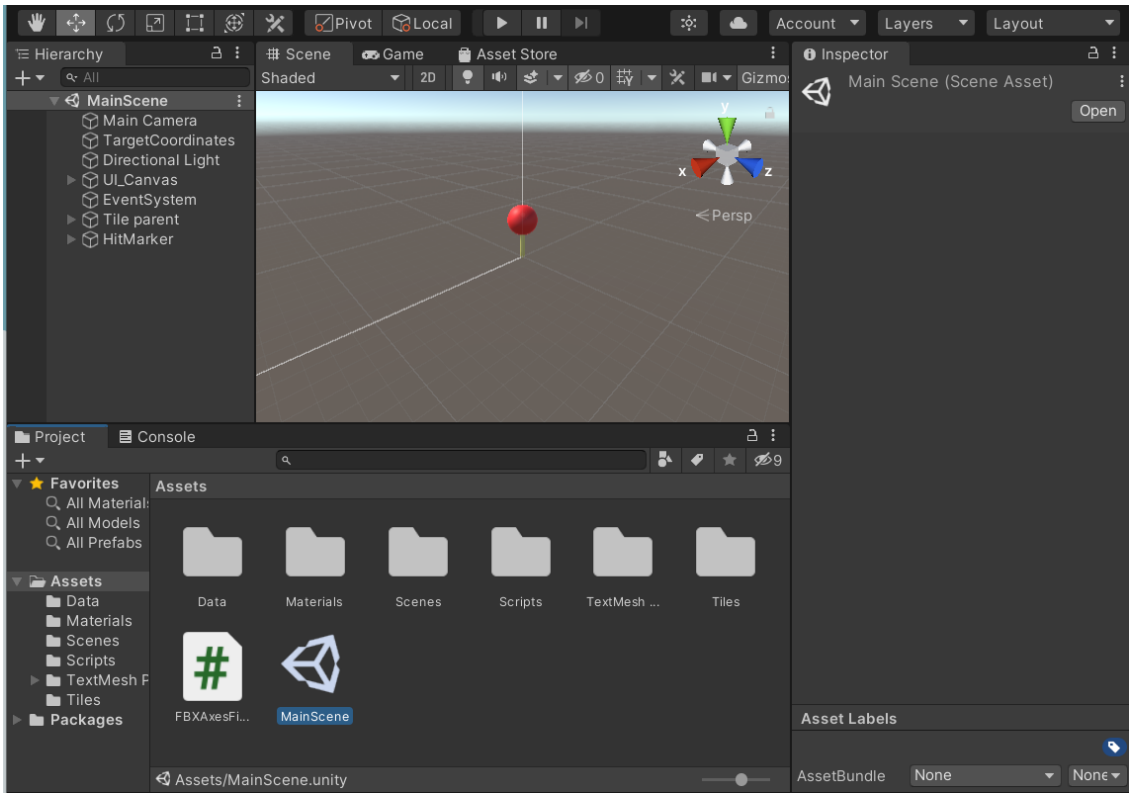

3. After clicking “ADD”, a window pops up where you locate the project folder that you have retrieved from GitHub. Select the project folder and click “Select Folder”.



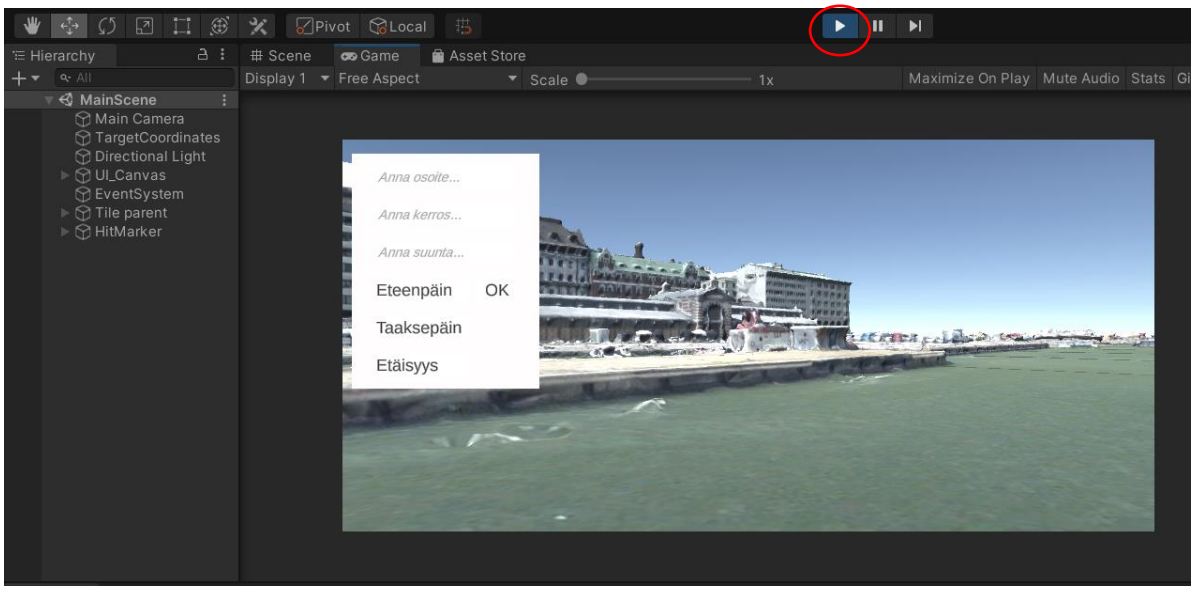

4. After selecting the folder, the project pops up into the Unity Hub view. Double click it to open in Unity.



5. Now the project should be opened in the Unity. Make sure that the “MainScene” is opened in the hierarchy. You can do this by double clicking the “MainScene”-file.




6. When you press the play button, it will run the project.



# Resources

## Unity 3D

Unity 3D is a game engine, which is used to create 2D and 3D games, animations, cinematics, architecture, engineering, construction and interactive simulations. The primary language of Unity 3D is C# programming language. This project uses Unity version 2020.03.0f1, which can be downloaded through Unity Hub or manually from their website. More information about the engine can be found at: https://unity.com/

## C# programming language

C# is a programming language developed by Microsoft, which runs in .NET system. C# is an object- and component-oriented programming language. It is used to develop web, desktop, mobile apps, games and other software.

## Data

### Address registry of Greater Helsinki region

The address registry covers the street names and other names in Finnish and Swedish for the Greater Helsinki region. The data is provided by the City of Helsinki through a WFS server as "Helsinki_osoiteluettelo". The address registry includes the coordinates of the addresses and the coordinate system is ETRS-GK25 (EPSG:3879). In this project the registry is in JSON format. More information about the data can be found at: https://hri.fi/data/dataset//helsingin-osoiteluettelo.

### Height data

The height data used for this software is an elevation model as an ASCII-file. Other types of file types are also supported if the row is constructed as: E N height. This software also supports worse resolution elevation models, like 5x5m, because it keeps track on the closest coordinate when it searches for the perfect match. The software is using 1x1m resolution of Helsinki open-source elevation model data as an ASCII filetype. More information about this data set can be found at: https://kartta.hel.fi/paikkatietohakemisto/pth/?id=256.

### 3D model

This software has a test area of 3D reality mesh model which represents Kalasatama, the district of Helsinki. The 3D reality mesh is divided in tiles and can be fetch from http://3d.hel.ninja/data/mesh/Kalasatama/. The test area is not covering the whole district of Kalasatama. In this software the used file format for the 3D model is FBX (Filmbox). The mesh model can be downloaded as zip files in 2km x 2km pieces. One piece contains 64 mesh model pieces at maximum and each of these smaller pieces cover the size of 250m x 250m. The pieces also come in a different level of detail, L13 is the lowest and L21 is the highest in detail. The level of detail can be seen from the name of the file e.g., Tile_+034_+028_**L21**_0033300. The test area used in the software has the L21 level of detail. More details of the 3D city model can be found at: https://hri.fi/data/en_GB/dataset/helsingin-3d-kaupunkimalli.

## Test area

The software does not cover the whole area of Greater Helsinki region. Instead of the whole region of the Greater Helsinki, the software uses 250m x 250m sized 3D mesh model (figure 6) of Helsinki and 1km x 1km sized height data area (figure 7).

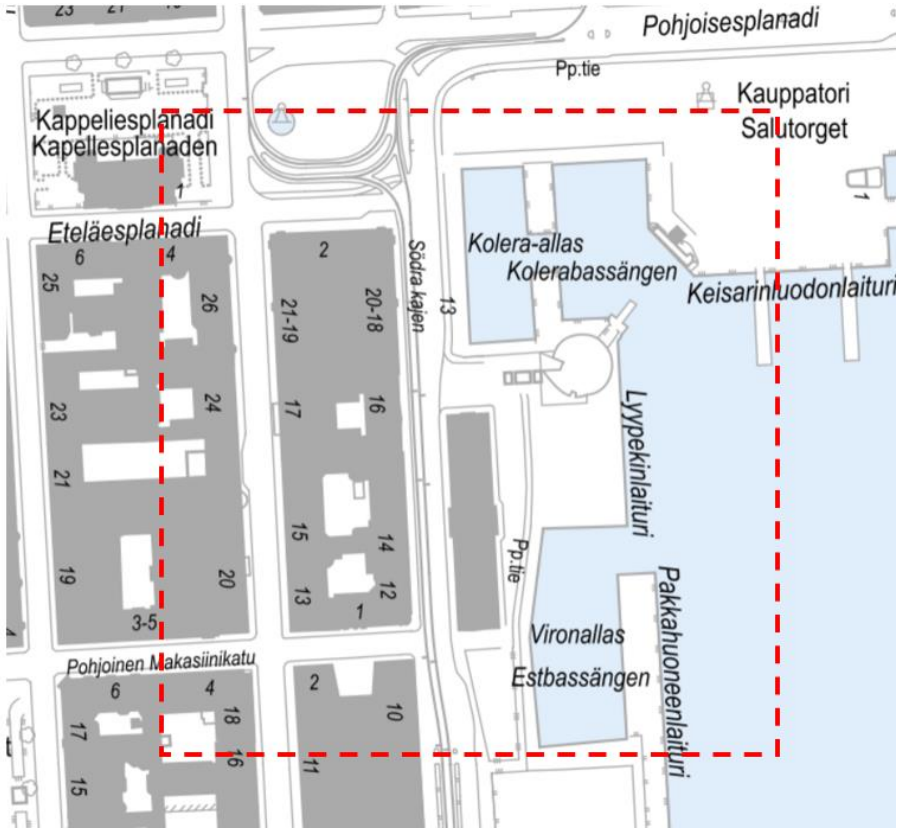



*Figure 6. The area marked in red dash lines represents the coverage of the 3D mesh model.*

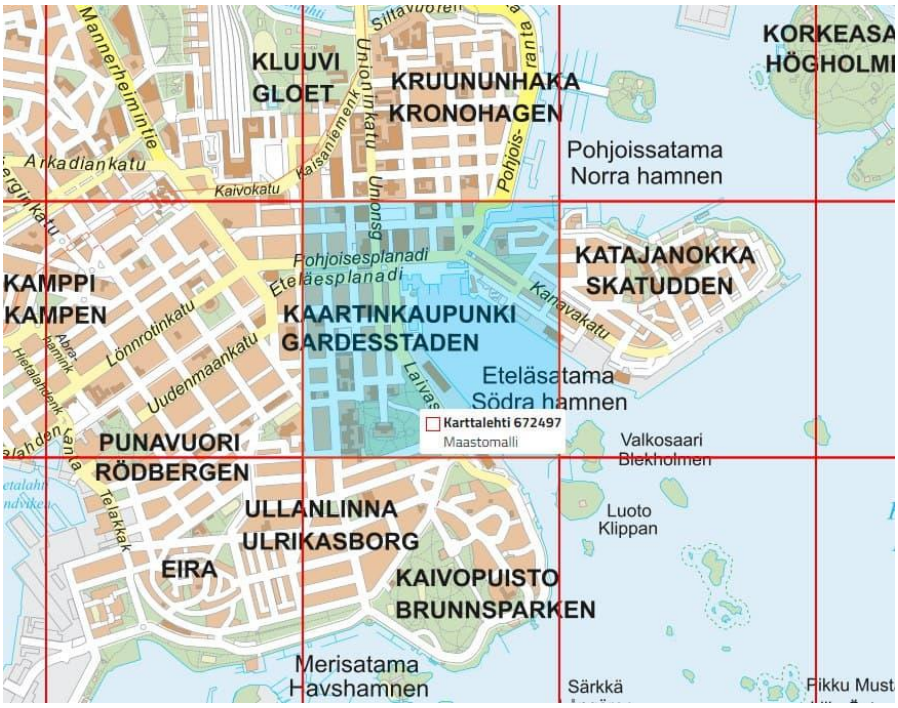



*Figure 7. The area marked in blue represents the coverage of the height data.*

## JSON parser

**Script file:** SimpleJSON.cs

**Location:** /Scripts/

**Developer:** Markus Göbel (Bunny83)

**Overview:**

SimpleJSON plusing is an easy to use JSON parser and builder. This plugin is called in AddressReader.cs script to parse the registry json file into better format. This makes it easier to use and search the registry for street addresses.

**Sources:**

https://github.com/Bunny83/SimpleJSON/blob/master/SimpleJSON.cs

http://wiki.unity3d.com/index.php/SimpleJSON

## FBX axes fixer

**Script file:** FBXAxesFixer.cs

**Location:** /Assets/

**Developer:** Sascha Graeff (FlaShG)

**Overview:**

The Y axis is vertical in the world coordinate system of the Unity while in the 3D mesh model the Z axis is vertical. FBXAxesFixer-script switches the Y and X axes of the 3D mesh model so it fits to the world coordinate system of the Unity. The script uses Unity Editor scripting class called AssetPostprocessor which works directly from the Asset files changing the output of the imported files.

**Sources:**

https://docs.unity3d.com/ScriptReference/AssetPostprocessor.html

https://gist.github.com/FlaShG/a82ae94d5789e92f2bec

## TextMesh Pro

TextMesh PRO is a Unity plugin for high-quality text. It is an easy way to add a professional touch because it contains many texts formatting options and appearance settings. More information about this plugin can be found at: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html.

# Scripts

## Using user defined parameters with button press.

**Script file:** ui.cs

**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> OK_button

**Public variables:** Input_Adress, Input_Height, Input_Direction, TargetEmpty, FloorHeight, CoordinateRealLife, CoordinateTransformationTarget, Layer, TestMode

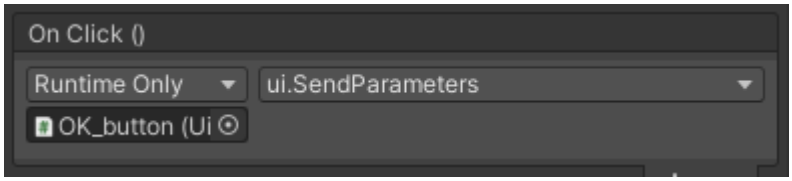
**Overview:**

When OK button is pressed, this script reads parameters from input fields to find the position for camera. The position is chosen to be facing away from a wall of a building in given address.

**Technical:**

Start method is called every time the program starts. In start method address reader and height reader are initialized for use in address search and height estimation.

SendParameters method is called every time OK_button is pressed. For this to happen, OnClick event for the OK_button is set to this method.



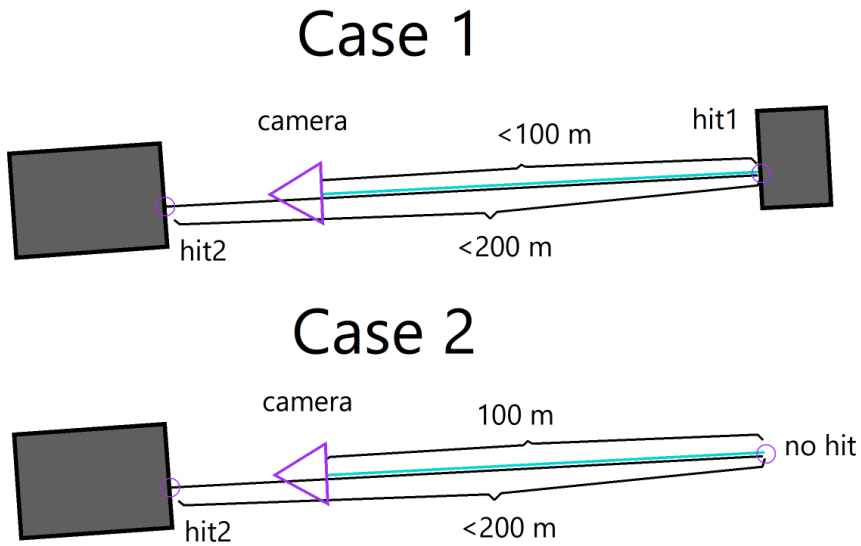

In SendParameters method, try-except format is used. This is to catch and ignore any errors caused by incorrect user input. In case of incorrect input, nothing happens in the standalone program. In Unity a debug log is printed.

Decimal and group separators are set for height and direction conversion from text field variables (Input_Adress, Input_Height, Input_Direction). Use "." as decimal separator on height and direction.

If the address given has changed from the last search, coordinates and height are searched using address reader and height reader. This process is somewhat slow, so searches are ran only when address changes.

The coordinates for x- and z-axes are given by address reader in ETRS89/ETRS-GK25FIN coordinate system and are transformed to fit coordinate system in Unity. The transformation happens using two public parameters: coordinateRealLife and coordinateTarget. These are coordinate vectors for a location in Helsinki that are used to calculate the difference between the two coordinate systems. As there is only translation (no scale difference, no rotation change) between the two systems, subtracting coordinateRealLife from coordinates of coordinateTarget.position will give the coordinate transformation.

These coordinates and the user specified direction are then used to find closest wall behind and camera target position (which the camera will follow, more about this on documentation of CameraMove script) is the moved there. Finding a wall is done using ray casting. First a ray at max length 100m is cast forward and then another ray back from where it hit a something (the mesh is defined as layer called tiles), or if nothing hits from 100 m away. The second hit is then used for camera location with extra 10 cm move forward to avoid clipping inside buildings.




For y coordinate, ground level is considered, as user input is just the floor, they want to be on. The following:

$$y\_coordinate = 1 + (height - 1) * floor\_height + ground\_height$$

Where height is the input and floor_height is how high each floor is (set to 3 m) and ground height is the height given by height reader for this location. Height value 1 is the street level and therefore one floor is subtracted from the input. The window height is estimated to be one meter above floor level.

## Find and return street address coordinates.

**Script file:** AddressReader.cs

**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> OK_button

**Public variables:** string inputAddress, string jsonFileLocation

**Overview:**

The script searches the registry of Greater Helsinki region and compares the user-defined street address to the registry's addresses. If a match is found the script returns the coordinates of the street address from the registry.

**Technical:**

The script consists of a callable function named returnCoordinates, which takes user-defined street address and registry file location as string parameters.

The input address parameter can be in lower- or uppercase and the function is able to identify redundant white spaces at the front and back of a string. The function is also able to detect whitespace that are between address and building number.

Regular expression also known as regex is used to identify street address, building number and letter from the input string. The identified objects are then grouped and stored into separate variables, which are used to search and match the data in the registry file.

Detecting the path of the registry file is semi-automatic. The folder of the project can be detected automatically through a Unity function. The registry file location string parameter is needed to find the file location inside the project folder. With this method reading the registry file is possible from any computer.

SimpleJSON node plugin is used to parse the registry file data, so that the registry can be searched for the input address. The search is done via a simple foreach loop that goes through all rows of the data. If the input address matches a row in the registry, then the function returns that row's coordinates in a list. If no match is found, then the function returns an empty list and prints a message to the Unity console that nothing was found.

**Notes:**

The printed message can only be seen at the Unity editor console, which means the message will not be visible if the project is built as a standalone application. Also, there is no fuzzy search, which means that the input street address has to match the address in the registry.

## Find and return coordinate height.

**Script file:** HeightReader.cs

**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> OK_button

**Public variables:** int[] inputCoordinates, string asciiFileLocation

**Overview:**

The script searches through an ascii-file for a height value by comparing the input coordinates with the file coordinates.

The script can be used with any text-file and not only with an ascii-type as long as the file consists of rows with the coordinates and heights. If an exact match for the input coordinates is found then it returns it, otherwise the script returns the closest coordinate height value.

The script returns the height as a float value.

**Technical:**

The script consists of a callable function named returnHeight, which takes user-defined coordinates as an int list and ascii file location as a string parameter. If the ascii file location is not in the same location as the script HeightReader.cs then it also needs the folder path. The ascii (or text file) consist of points, which is in the format of: *E N height*. The east coordinate needs to be before the north in the text row.

The script is searching through each row in the ascii file for the correct coordinates. It does also keep track for the closest height value, until the correct coordinates are found. If the correct coordinates are not found from the file, the script returns the closest height value.

The closest height value is calculated by calculating the distance between input and file row coordinates using Pythagorean theorem. The length of the two sides in the triangle in Pythagoras theorem is received by taking the difference of the X coordinates and Y coordinates. By using the formula below, we can calculate a difference between each point.

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

When a new closer distance is found the old closest distance and height is then overwritten with the new distance and height. And by calculating this distance for every row while the correct coordinates have not emerged, we can keep track of the closest if the exact correct coordinate is not found.

Because different cultures use different decimal separators the script need to be aware of how to deal with points and dots when converting strings to a numerical shape. This script tries to convert the numerical strings into int and float formats and if it is not able to do it the script removes the original decimal separator and replaces it with a point before it moves on to comparing the coordinates.

**Notes:**

The printed message can only be seen at the Unity editor console, which means the message will not be visible if the project is built as a standalone application.

This script will only search for the height in one elevation model file and return the closest height. To be able to search through multiple files for the correct height this script needs some changes. This is a possible implementation in the future.

## Moving the position of the camera to the street address coordinates.

**Script file:** CameraMove.cs

**Location:** /Scripts/

**Component of:** MainCamera

**Public variables:** target (transform), Freemode (Boolean), smoothTime (float), offset (Vector3), field1-3 (input fields)

**Overview:**

Each frame the program is running this script either moves the camera to target location or allows for free movement depending on if Free Mode is activated. Using WASD keys and mouse the user can move freely in the scene when Free Mode is activated by pressing “f”.

**Technical:**

If user interface is active (any of the input text fields is active), variable UI_active is changed to true. When UI is not active, it is possible to activate and deactivate Free Mode by pressing “f” (value of boolean FreeMode changes).

If Free Mode is not active, camera is moved smoothly towards location and rotation defined by empty object defined in variable target. Also, if UI is not active, WASD key movement is possible.

If Free Mode is active, in addition to the script enabling moving the camera horizontally (X and Z axis) with WASD keys relative to the direction of the mouse the mouse can be used to control the direction that is viewed. The variable “lastMouse” takes the user’s current mouse position in pixel coordinates as input, and then the input is being transformed as new rotation in the world space. In other words, it defines the camera angle by user’s mouse movement. The horizontal movement happens through the “WASDKeyMovements”-function which returns the values of the target positions made with WASD keys. How it works is that “Input.GetKey()” returns true while the user holds down one of the WASD keys. When it is true, the if-command inside the function adds (0,0,1), (0,0,-1), (1,0,0) or (-1, 0, 0) to the “p_WASD”-vector according to the pressed key. Then “p_WASD”-vector is assigned as a new vector and it is multiplied to speed up the movement.

## Measuring distance to the furthest visible point.

**Script file:** MeasureDistances.cs

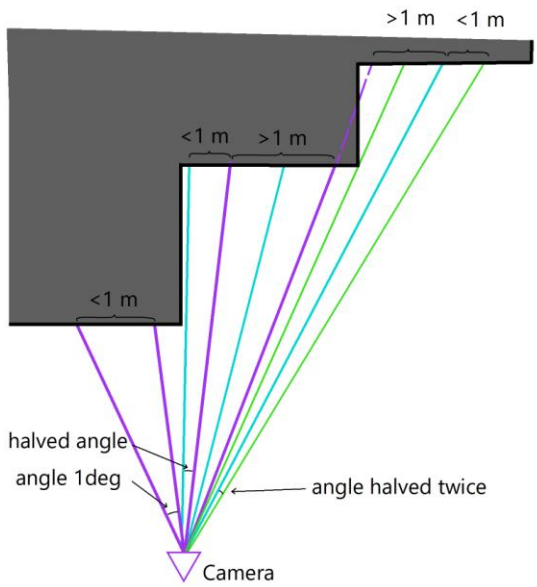
**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> Measurements

**Public variables:** Camera: Main Camera, Layer: Tiles, Marker: HitMarker, DistanceText: UI_Canvas -> Menu -> DistanceText

**Overview:**

This script uses raycasting to measure distance to the furthest point in the view. Rays are casted in one-degree intervals. If set view angle between two rays is more than one meter where the ray hits,



the angle is halved, and new rays are cast adjacent to the previous one. This is continued recursively. The results are shown with a pin on the view and a distance label above the pin.

**Technical:**

The field of view is set for vertical and horizontal directions. These values are used for defining the width of the whole raycasted area. The raycasting will start from the upper left corner and go row by row with one degree angle increments in a for-loop through the set field of view.

If the ray hits a tile, distance to hit is checked. The distance between two neighboring hits at this distance is calculated using

$$neigbouring\ distance = 2 * \tan\left(\frac{\pi}{180}\right) * angle * distance$$

and if it is more than one meter, more rays will be cast above, below, right, and left of the previous ray with the angle between rays decreased to half. For these following rays, the same check is done, and new rays casted recursively if the distance between neighboring rays angles is larger than one meter. The recursion happens in CastRay method. When the desired below one-meter accuracy is achieved, the ray with largest hit distance value is returned and compared to the furthest distance so far. The current furthest hit ray is stored in variable named furthest.

Recursively casting rays with smaller angle between happens with CastRay method. Here any given previously casted ray and angle between neighboring rays are used to cast more rays closer to this previous ray. The new rays are cast in four directions: up, right, down, and left. Then for each of those new rays hit distance is checked and the distance to neighboring ray. If distance to neighboring ray is more than one meter, the same process continues. If the distance is less than one meter, the distance is checked against furthest raycast hit distance so far and returned if it is further.

**Notes:**

For each ray that hits, Debug.DrawRay is called to visualize the ray in the Unity editor Scene view. This is a debug feature, that is only visible in the editor, not in the final product.

The code on this part works but is not the most elegant, read at your own risk 😉.

## Show distance on top of pin showing furthest visible location.

**Script file:** DistanceText.cs

**Location:** /Scripts/

**Component of:** HitMarker -> TextPosition

**Public variables:** DistanceLabel: Ui_Canvas -> Menu -> DistanceText (Label)

**Overview:**

Text label showing distance to the furthest visible point in view is moved on top op of the pin showing this location.

**Technical:**

Text label defined with distanceLabel variable is moved every frame to be on top of the pin. The position is defined by HitMarker and its child object (empty) TextPosition, which is used to offset the text a little to not be in front of the marker.

## Move the camera to the building at the front or back.

**Script file:** ChangeBuildingSide.cs

**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> Forward, Backward

**Public variables:** targetEmpty, layer

**Overview:**

The purpose of the script is to move the camera forward or backwards one building, while retaining current orientation.

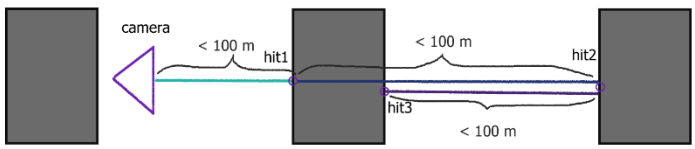
**Technical:**

The script has two functions, which are RaycastForward and RaycastBackwards.

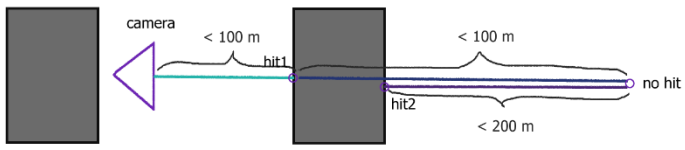
RaycastForward()

A ray is cast forward from the location of the camera. If the ray hits something within 100 meters, cast a second ray at the same direction from first hit coordinates with 10cm offset. If the first ray hits nothing, then print a message to the editor console. If the second ray hits something within 100m, cast a third ray backwards from where the ray came. If the second ray hits nothing within 100m, cast a third ray back from 100m away. In both cases the third ray hit coordinates are used to move the camera to the coordinates location with 10 cm forward offset to prevent camera from clipping to the inside buildings.

Case 1



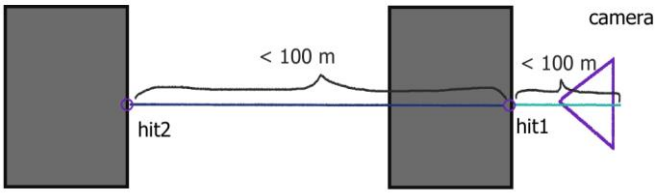

Case 2




RaycastBackward()

A ray is cast backwards from the location of the camera. The first ray should hit the building behind the camera. If the ray hits, cast a second ray to the same direction from the hit coordinates with

slight offset. If the first ray does not hit anything, print a message to the Unity editor console. If the second ray hits something move the camera to the hit coordinates with 10 cm forward offset to prevent camera from clipping to the inside buildings.




## UI move to next field with tab.

**Script file:** NextInputField.cs

**Location:** /Scripts/

**Component of:** UI_Canvas -> Menu -> Input_Adress, Input_Height, Input_Direction

**Public variables:** Next Field: Input_Adress or Input_Height or Input_Direction (TMP input field)

**Overview:**

When user has input field active, pressing Tab will change the active field to next one. Same script is used for every input field.

**Technical:**

Each frame the script checks if the input field this script is component of is active and TAB button is pressed. If this is true, input field given as NextField variable is activated which also deactivates current field.

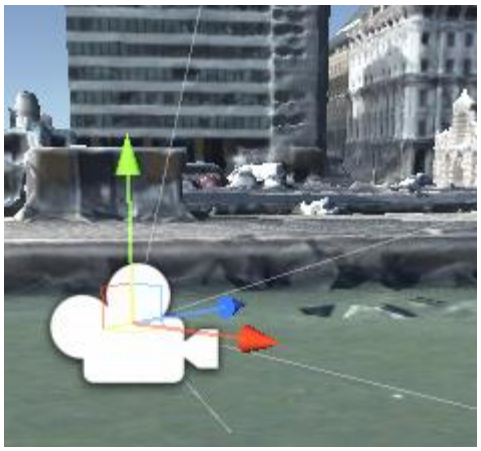
# Game Objects

## Helsinki Scene:

All game objects are under a scene, called Helsinki Scene.

**Location:** /scenes/

### Main Camera:

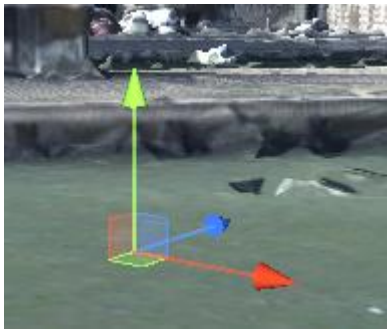


The project has only one camera called Main Camera. The camera has default position at launch, which can be changed by moving it in the scene. Camera follows frame by frame target coordinates (empty object) according to CameraMove script. If camera FOV is changed, FOV's in MeasureDistances-script should be changed correspondingly.

**Scripts:** CameraMove.cs

**Location:**

### Target Coordinates:



Target coordinates for the camera. When target moves, camera follows if Free Mode is not active.

**Scripts:** None

**Scripts that can move the object:** ui.cs, ChangeBuildingSide.cs, CameraMove.cs

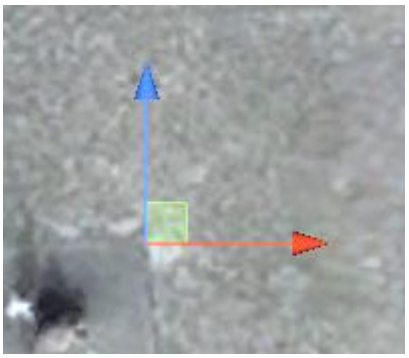
### Directional light:

Object defines the direction from which the lighting comes from. Location and coordinates (0,0,0), but does not matter where it is located.

**Scripts:** None

## Tile parent:

Tile parent is an empty object, under which each tile used in the mesh model are put. Changing the location of tile parent will change location of everything under it. Under tile parent is also empty object called CoordinateTransformationTarget, which is used to transform ETRS89/ETRS-GK25FIN used by City of Helsinki to coordinates used in Unity engine. The location where the empty object is in reals life is measured at (25497471.56, 6672725.34) which is set as variable in ui.cs. As CoordinateTransformationTarget is under Tile parent, the coordinate transformation can be calculated in ui.cs with any location for the mesh model in the scene.

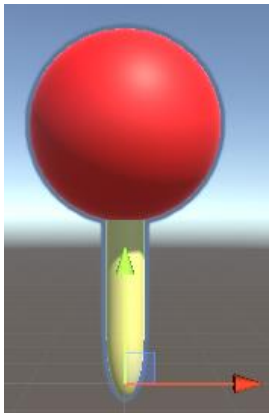


**Scripts:** None

**Scripts that use CoordinateTransformationTarget:** ui.cs

**Location:** /tiles/

## HitMarker:



HitMarker is the pin model parent object, that is moved to the visible location which is furthest from the camera when measuring distances. The marker consists of two child 3D model parts (ball and leg) and an empty object that is used to offset distance text from the 3D model. The distance text shows distance from camera to the tip of the pin and is part of UI_canvas.

The colors of the leg and ball are materials set in the /materials/ folder.

**Scripts:** None

**Scripts that can move the object:** MeasureDistances.cs

## Menu / UI

The menu is used in operating the software. The menu consists of three input fields and four buttons, of which one is the OK button that is used to operate the inputs. In addition to these input fields and buttons that are explained in this chapter the software also uses specific keyboard buttons to move the camera around the model. More about these keyboard buttons in the Moving the view by WASD keys and mouse chapter. Tab keyboard button can be used to move between the input fields. More about this in UI move to next field with tab chapter. In figure 8 the layout of the menu is visible.

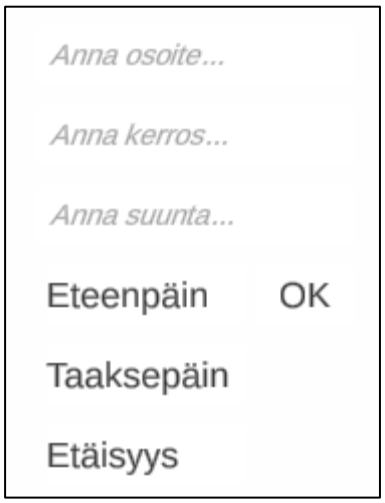



*Figure 8. the Menu / UI*

### Anna osoite…

This input text field is used to give the software the wanted address location that the user wants to look at the window view from. The address needed for the input is from Address registry of Greater Helsinki region. More about this data set in the Data chapter. The address input can be upper- or lowercase but needs to match the address exactly correct. I.e., there is no fuzzy search implemented. The input cannot either be a non-existing street number. More about the address search in the Find and return street address coordinates chapter.

### Anna kerros…

This input field is used to give the software the wanted building floor, from which the user wants to look at the window view.

To be able to move the camera to the window level on the correct floor the software needs to know the height above sea level, the wanted floor, and the height of a floor. The first is computed, the second is user-given and the last is assumed. The target height is calculated using the formula in Using user defined parameters with button press chapter where the target height is the y-coordinate.

The height above sea level will be computed using the received coordinates from Address reader script, which will be sent to the Height reader script which will give the software the height above the sea level. A closer description about these processes can be found in the chapters: Find and return street address coordinates and Find and return coordinate height. The wanted floor is a user-given input string from this input field that will be transformed into a numerical value.

The floor height in Finland is regulated to be a minimal of 3.0 m (Ympäristöministeriön asetus asuin-, majoitus- ja työtiloista 1008/2017). Using this regulation, we make the assumption that the most used floor height in Helsinki is 3.0 m, and this is the value the software is using for the floor height.

### Anna suunta…

This input field is used to give the software the wanted direction of the window view. The input is in degrees and the corresponding air directions and degrees can be seen in the figure below. Basically, North is 0 degrees and then it rotates clockwise so that 360 degrees is also North. The camera then rotates to the right direction when moving to the address location. More about his in the Using user defined parameter with button press.

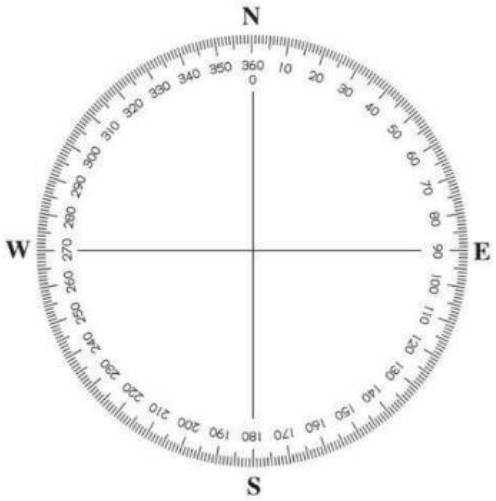



*Figure 9. Figure to explain the relation between air directions and degrees.*

### OK

By pressing this button, the software receives the input strings from the input fields and moves the camera to the right location. More about his in the Using user defined parameter with button press.

### Eteenpäin and Taaksepäin

These buttons are used when pressed to move the camera to the following or the previous buildings after the user have given their initial input address, height, and direction. The Eteenpäin-button moves the camera forward to the following building in the same view direction that was given by the user. The Taaksepäin-button moves the camera backward to the previous building in the opposite direction as the user-given direction while still maintaining the view direction. The camera moves always so that the camera view is outside of the building. A more detailed description of these button functions can be found in the Move the camera to the building at the front or back chapter. These buttons can be used for example to move in and out from the inner ward of the buildings.

### Etäisyys

This button computes when pressed the location of the viewpoint that is furthest away from the camera location and gives the distance to it. When the furthest distance is found the software places a pin in that location and prints the distance to it above the pin. A detailed description about this functionality can be found in the Measuring distance to the furthest visible point chapter.

# Possible improvements to consider for the future development:

## Improvements in the code:

- MeasureDistances.cs code has lots of repetition which could be removed by calling as function.
- Adding Fuzzy Search for AddressReader.cs
- Improving efficiency of search algorithms for both AddressReader.cs and HeightReader.cs
- Improve HeightReader.cs so that the script would be able to use multiple elevation model tiles.

## Improvements in 3D model:

- Adding asset streaming to avoid downloading and packaging every 3D mesh tile into the software.

## Other:

- Combining with CityGML model to get semantic information about buildings such as actual window locations.

# Sources


Ympäristöministeriön asetus asuin-, majoitus- ja työtiloista 1008/2017. Given in Helsinki 20.12.2017. Available at https://finlex.fi/fi/laki/alkup/2017/20171008.

Unity 3D https://unity.com/

Helsinki 3D mesh model https://hri.fi/data/en_GB/dataset/helsingin-3d-kaupunkimalli

TextMesh Pro https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

FBX asset fixer https://gist.github.com/FlaShG/a82ae94d5789e92f2bec

JSON Parser https://github.com/Bunny83/SimpleJSON/blob/master/SimpleJSON.cs

Height data https://kartta.hel.fi/paikkatietohakemisto/pth/?id=256

Address data https://hri.fi/data/dataset//helsingin-osoiteluettelo