Data Generated Dashboard

Using Data from Grasshopper to Generarte UIs in Unity

https://github.com/keijiro/GeoVfx

SHANAIRE BLYTHE

17th March 2023

Objectives For the Session

1 Exporting Data From Grasshopper to CSV

2 Reading CSV Data in to Unity

3 Create a Dashboard from Data

4 Add Basic Interection

5 Exporting Unity Model to WebGL (Convert Project Over)

Resources

https://resources.shanaireworks.co.uk

Link to download resources

Name: Shanaire Blythe

Qualification: MArch Architecture

Occupation: Software Developer

Interests: Concept Art, VFX, Drawing, Martial Arts and more.

Name: Shanaire Blythe

Qualification: MArch Architecture

Occupation: Software Developer

Interests: Concept Art, VFX, Drawing, Martial Arts and more.

//COLLABORATE

//Shanaire Blythe

with Adam Brennan and Kit Yeung Chan

CPU - M.Arch

SITE ANALYSIS

Investment and Amenities

By drawing maps consisting of residential property prices, vacant plots and amenities such as schools and parks, we aimed to understand how investment in East Manchester is affected. As the current regeneration schemes are focused on residential development we looked at house prices and how they differ based on their location in Manchester as a whole and their distribution in relation to the amenities in East Manchester.

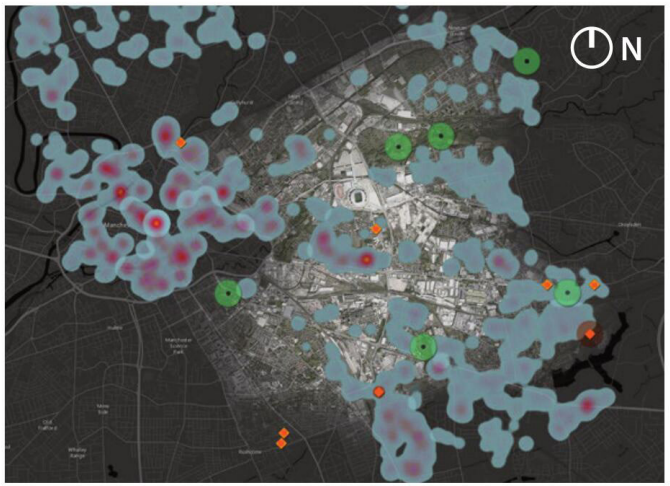
As can be seen by the colour scale, East Manchester's properties fall into the lower end of the price spectrum. When compared with the city centre this price drop is very apparent and the further into the area houses are, the lower the prices seem to be. An expected result is that prices would be marginally higher for areas located near parks and schools.

Park

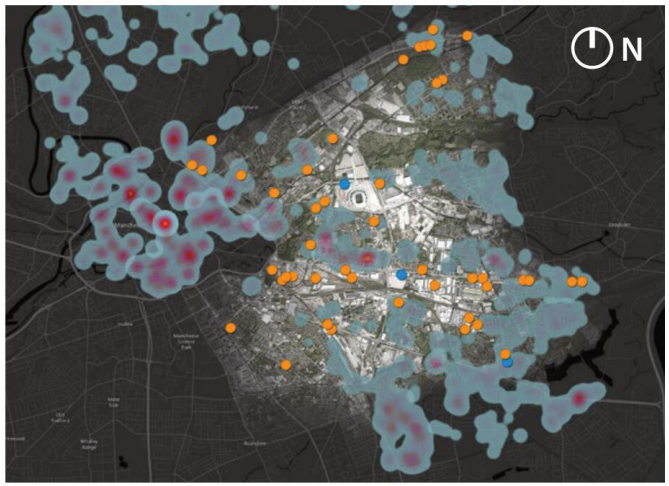
Scale of school

Unoccupied site

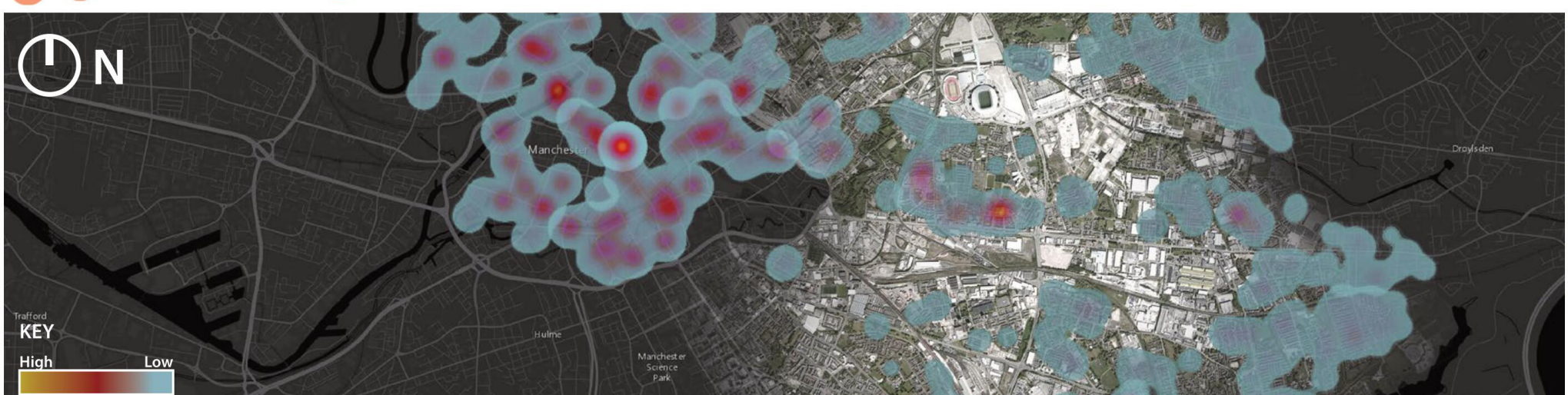
Unoccupied building



Schools, Parks & Properties Price



Unoccupied sites & Properties Price



PANARCHY

East Manchester

Applying the panarchy to East Manchester, we can use it to represent three different scales in society which are the Macro, Meso and the Micro scales. We will relate these three scales to:

Regeneration Planners - Goverment level actors who are able to make policy decisions and top-down effects on East Manchester.

Developers - Any non-goverment actors who transform the urban morphology through land development and building.

Citizens - Residents of East Manchester who are also non-government.

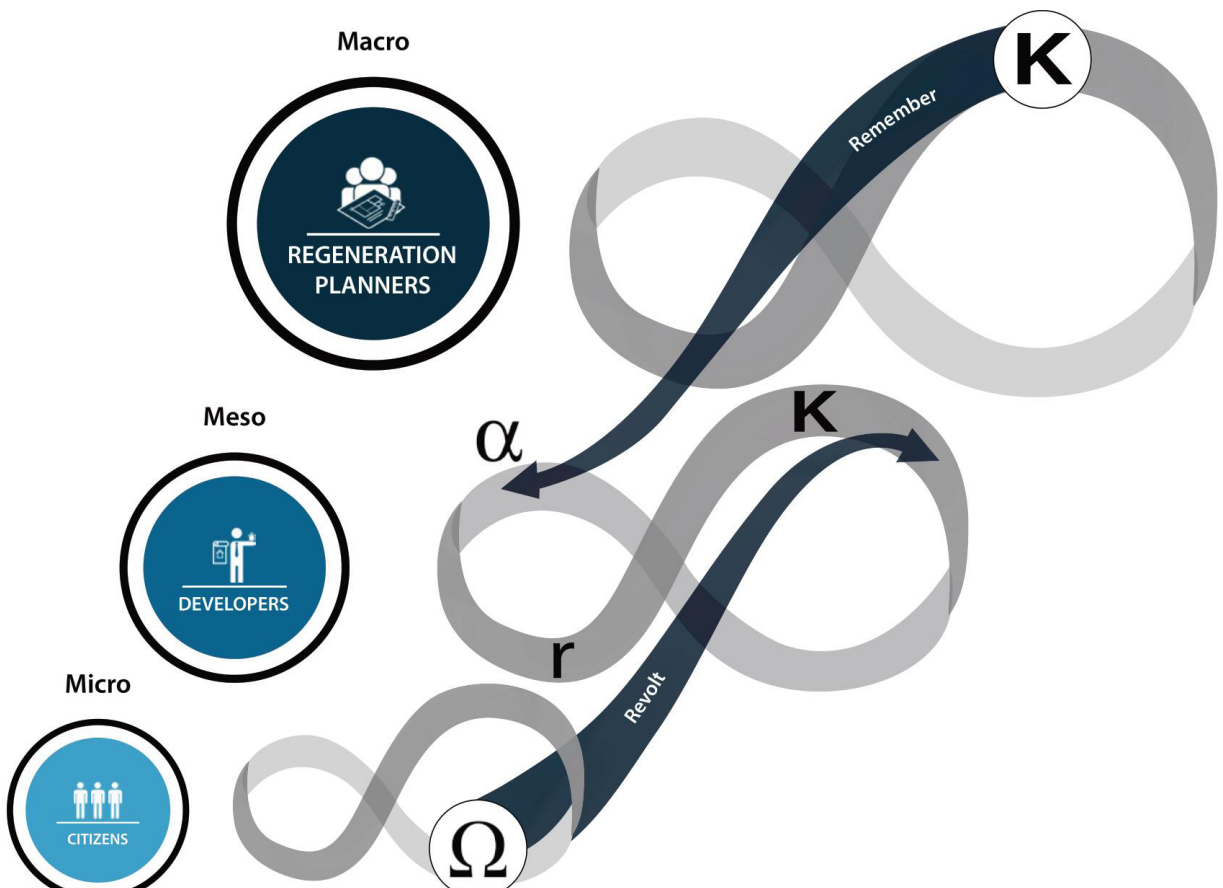
Rather than the city being looked at like the typical hierarchical top-down structure, the panarchy views these scales as different cycles where there is a level of interaction between them rather than as separate entities in total isolation from other levels in a hierarchy. The three groups are representative of organisations or groups that are around their size in society. Using this approach gives us the opportunity to explore interactions between levels, which is crucial to our approach to create better interaction between them.

By assuming a city evolves across different levels and at different speeds this sets out requirements for the proposed digital tool:

In terms of users, it implies that feedback should come from agents at different levels in order to best represent adaptation.

The model must also record data and interactions between the levels in order test links and thus integrate or reject them.

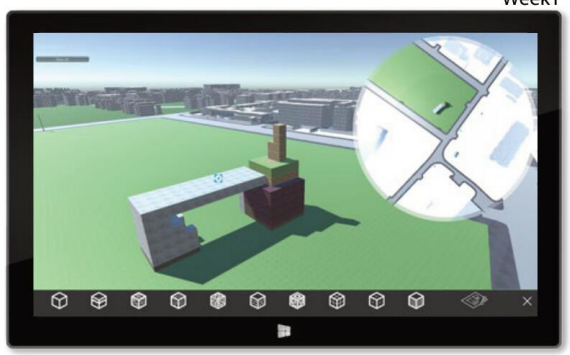
It should allow individual actions combine together to make large patterns, such as the movements of people, information, goods or money.



OUTCOME

Possibilities

Week1



Week2

Week3

Week4

?


COLLABORATE
Creation

Name: Shanaire Blythe

Qualification: MArch Architecture

Occupation: Software Developer

Interests: Concept Art, VFX, Drawing, Martial Arts and more.

MF3D

Architectural Visualisation

Bonded Warehouse
18 Lower Byrom Street
Manchester
M3 4AP

MF3D

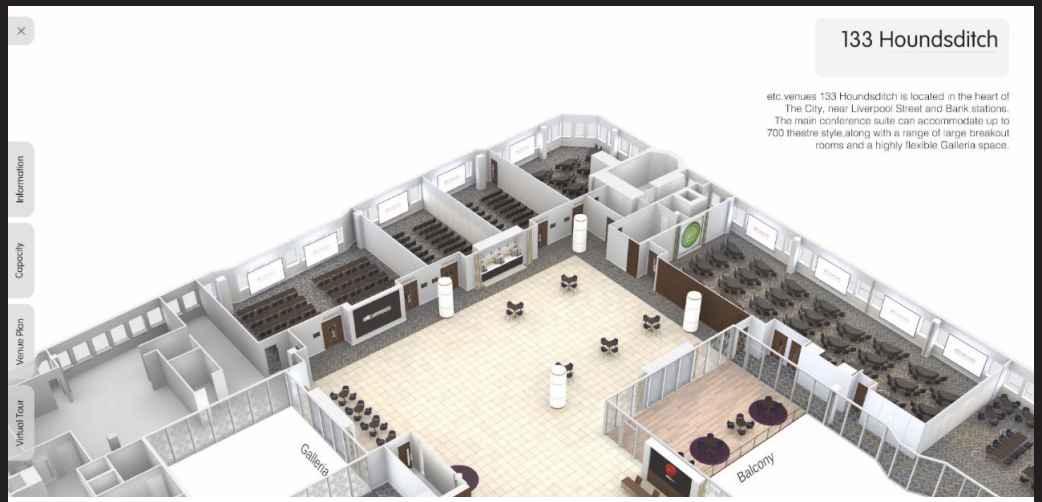
Working for 3 years +

Bonded Warehouse
18 Lower Byrom Street
Manchester
M3 4AP

MF3D

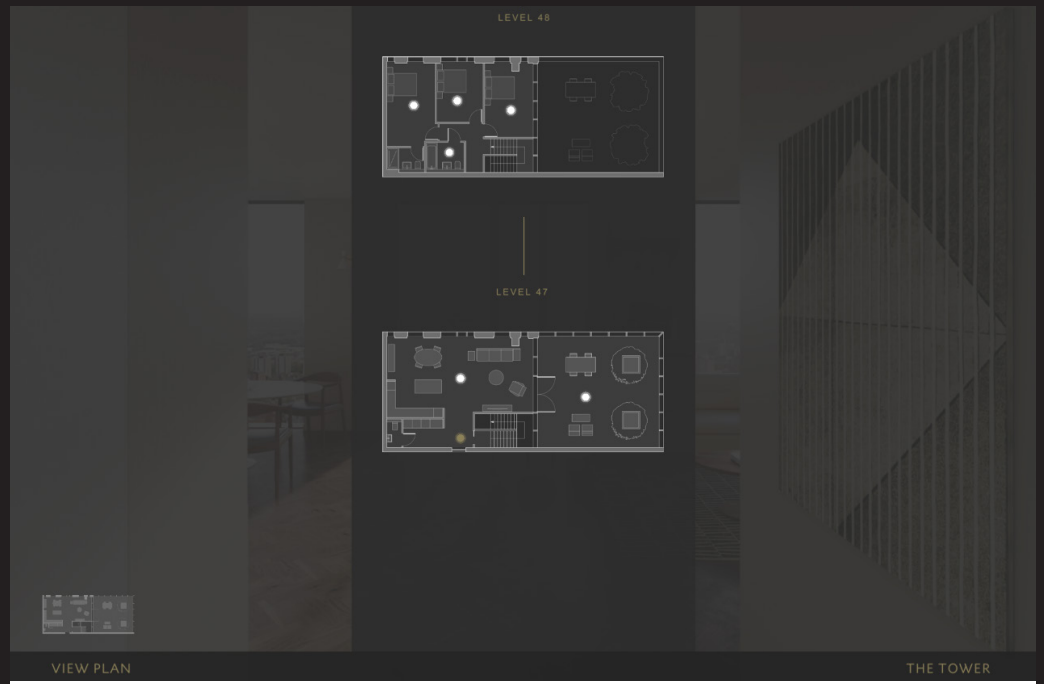
Creating Interactive Applications

Bonded Warehouse
18 Lower Byrom Street
Manchester
M3 4AP



133 Houndsditch 3D Floor Plan - Space Planner

Bonded Warehouse (Android / iPad App)



One ST Johns

3D Floor Plan Static Web Application

Name: Shanaire Blythe

Qualification: MArch Architecture

Occupation: Software Developer

Interests: Concept Art, VFX, Drawing, Martial Arts and more.

Exporting Data From Grasshopper

CSV

```
Date,Price
01/01/2020,1741005
01/02/2020,1760434
01/03/2020,1730654
01/04/2020,1650120
01/05/2020,1643456
01/06/2020,1530076
```

CSV files are expressed by the comma that separates the column of data, with each line representing a different data record.

JSON

```
{
    "widget": {
        "visible": "on",
        "window": {
            "title": "Sample Konfabulator Widget",
            "name": "main_window",
            "width": 500,
            "height": 500
        },
        "image": {
            "src": "Images/Sun.png",
            "name": "sun1",
            "hOffset": 250,
            "vOffset": 250,
            "alignment": "center"
        },
        "text": {
            "data": "Click Here",
            "size": 36,
            "style": "bold",
            "name": "text1",
            "hOffset": 250,
            "vOffset": 100,
            "alignment": "center",
            "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
        }
    }
}
```

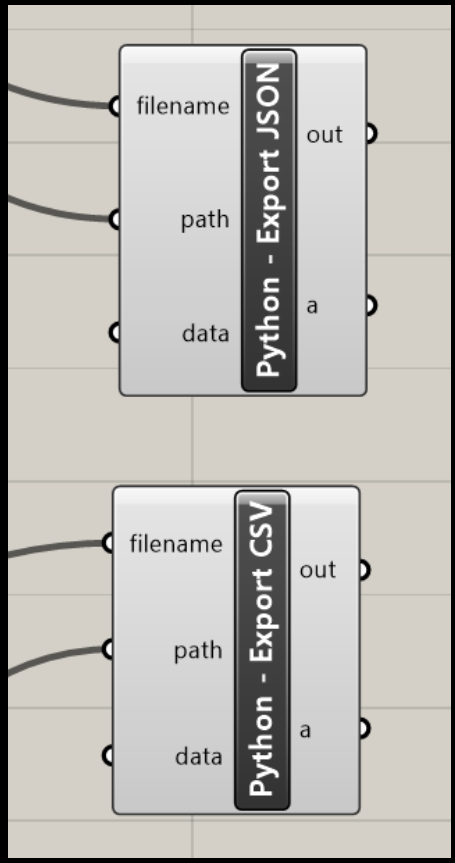
JSON is a text format that is used to represent structured data, which is based on how JavaScript Notation, using key value pairs.

CSV & JSON

To save the data from Grasshopper we will be using the python node. This will be used to fomat and save the data as either CSV or JSON.

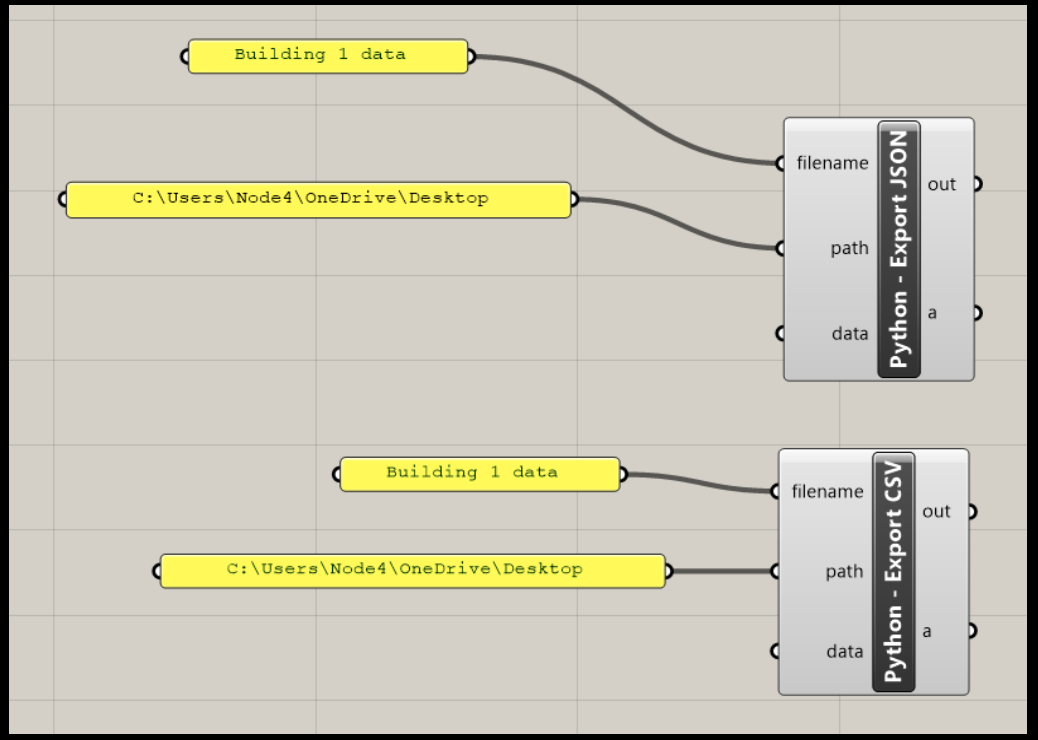
Instructions:

+ Create Python Node
+ Add the following input variables
    - filename = Name of the file (panel)
    - path = Path Save folder (panel)
    - data = data to be saved (gh data)



CSV & JSON

For this example, we will be using the desktop to store the data.



CSV

Creating a CSV file will be much simpler than JSON, for this all we will need to import is the “os” library, which gives us access to the ability to controlling different file path on the system

Instructions:

+ Import os library
+ Create a main function to control execution order
+ Create a csvData variable, which specifies the first row of the csv file

```
10
11  import rhinoscriptsyntax as rs
12  import os
13
14  csvData = "index, height"
15
16  def main():
17      pass
18
19  main()
```

JSON

A JSON file follows a similar structure to a dictionary in python. So we will be working with the data as a dictionary, then save it to JSON afterwards.

Instructions:

+ Import os library
+ Inport json library
+ Create a main function to control execution order

```
import rhinoscriptsyntax as rs
import os
import json

"""
{
    "name": "_name",
    "height":[
    1,2,4
    ]
}
"""

def main():
    
    pass

main()

```

Demo

CSV

Example code for
CSV export

```python
import rhinoscriptsyntax as rs
import os
import math

def SaveCSV(p_dir, p_folder, p_filename, p_csvData):
    if (os.path.exists("{0}/{1}".format(p_dir, p_folder))):
        with open("{0}/{1}/{2}.csv".format(p_dir, p_folder, p_filename), "w") as file:
            file.write(p_csvData)
    else:
        os.mkdir("{0}/{1}".format(p_dir, p_folder))
        SaveCSV(p_dir, p_folder, p_filename, p_csvData)

def StructureCSV(p_data, p_csvData):
    for index, item in enumerate(p_data):
        p_csvData += "\n{0},{1},{2}".format(index, item, math.floor(item))
    return p_csvData

def main():
    csvData = "index, height, int height"
    folder = "ghData"
    
    os.chdir(path)
    currDir = os.getcwd()
    csvData = StructureCSV(data, csvData)
    SaveCSV(currDir, folder, filename, csvData)

main()
```

JSON

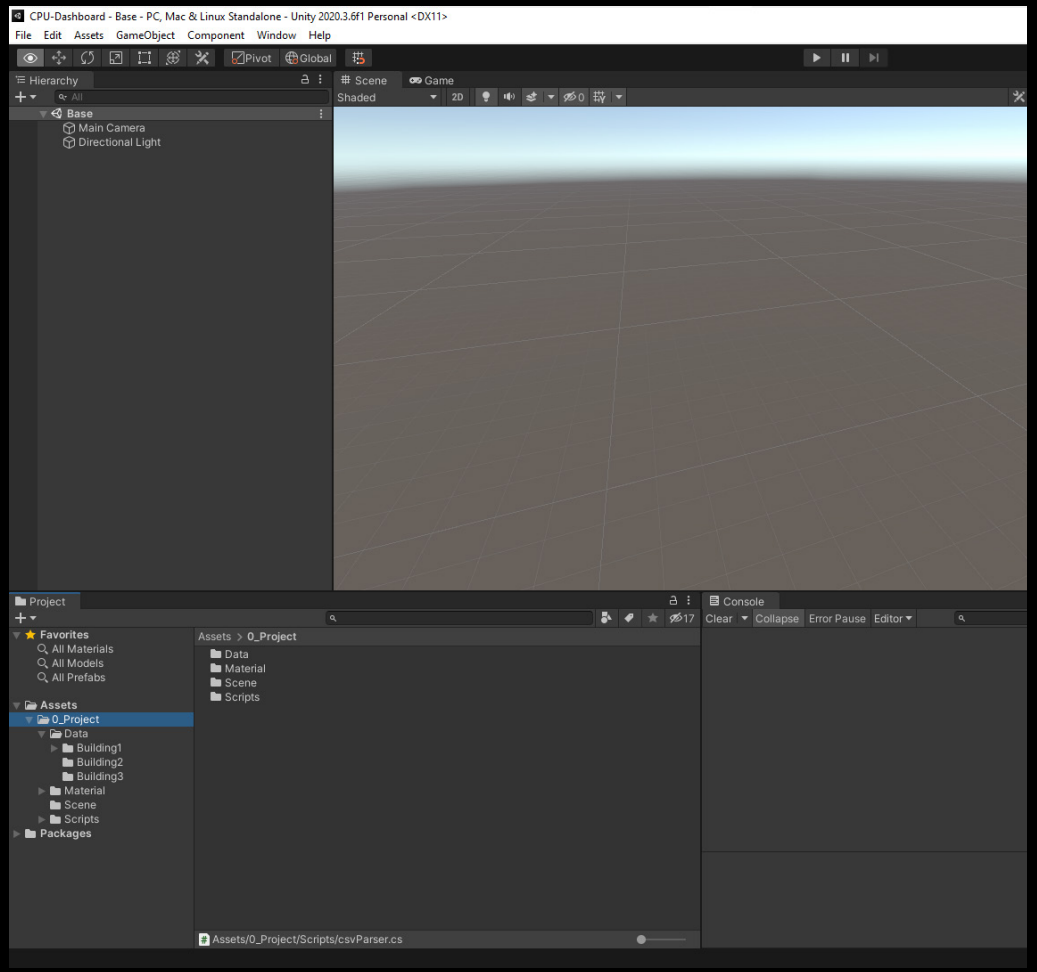
Example code for
JSON export

```
import rhinoscriptsyntax as rs
import os
import json

"""
{
    "name": "_name",
    "height":[
    1,2,4...
    ]
}
"""

def SaveJSON(p_dir, p_folder, p_filename, p_json):
    if (os.path.exists("{0}/{1}/".format(p_dir, p_folder))):
        with open("{0}/{1}/{2}.json".format(p_dir, p_folder, p_filename), "w") as file:
            file.write(json.dumps(p_json))
    else:
        os.mkdir("{0}/{1}/".format(p_dir, p_folder))
        SaveJSON(p_dir, p_folder, p_filename, p_json)

def StructureJSON(p_Data):
    jsData = json.loads("{}")
    jsData["name"] = "building data"
    jsData["height"] = p_Data
    return jsData

def main():
    folder = "ghData"
    
    os.chdir(path)
    currDir = os.getcwd()
    
    json = StructureJSON(data)
    SaveJSON(currDir, folder, filename, json)

main()
```

Read Exported Data Into Unity

Create a CSV or JSON Parser

To load the exported data or any data that you have on file, we will need to write a parser so that unity can understand the structure and fields.



Instructions:

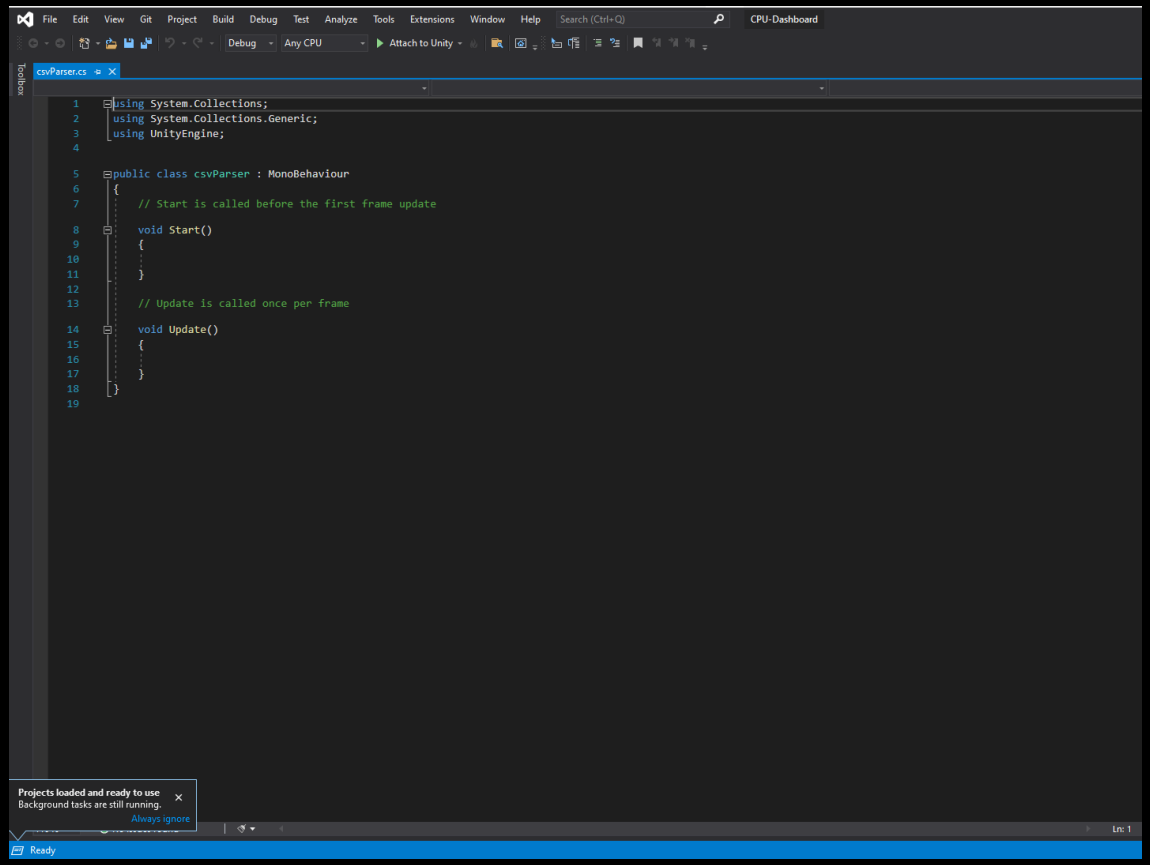
+ Open Unity
+ Create a blank Scene & Load it
+ Create a Simple File Structure
    - 0_Project
        - Data
        - Material
        - Scene
        - Script

Create a CSV or JSON Parser

Now you can add your data files into the project inside of the data folder.

Inside the **scripts folder** create a file called **csvParser** and open the file in your text editor of choice.

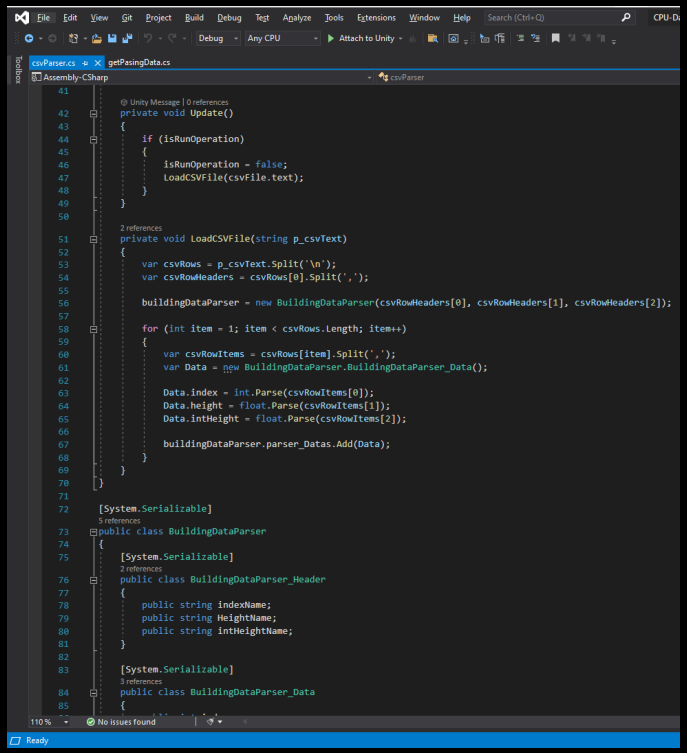
One thing to remember is that we want the data to only load at the **Start of the Application**



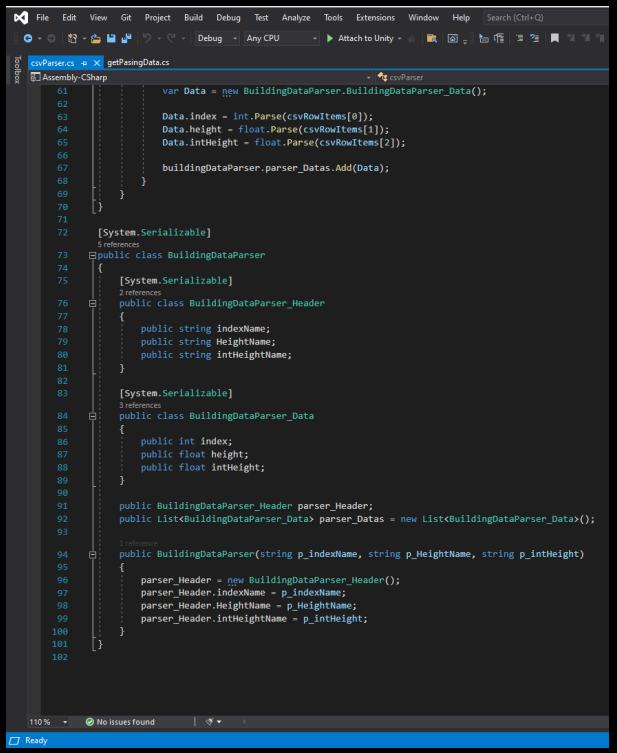
Demo

Create a CSV Parser

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[ExecuteInEditMode]
Unity Script | 4 references
public class csvParser : MonoBehaviour
{
    private static csvParser Instance = null;

    1 reference
    public static csvParser GS_Instance
    {
        get
        {
            if (Instance == null)
            {
                Instance = FindObjectOfType<csvParser>();
            }
            return Instance;
        }
    }

    1 reference
    public BuildingDataParser G_GetBuildingParserData
    {
        get
        {
            return buildingDataParser;
        }
    }

    [SerializeField] TextAsset csvFile;
    [SerializeField] BuildingDataParser buildingDataParser;
    [SerializeField] bool isRunOperation;

    Unity Message | 0 references
    void Start()
    {
        if (buildingDataParser == null)
        {
            LoadCSVFile(csvFile.text);
        }
    }

    Unity Message | 0 references
    private void Update()
    {
        if (isRunOperation)
        {
```



```
Unity Message | 0 references
private void Update()
{
    if (isRunOperation)
    {
        isRunOperation = false;
        LoadCSVFile(csvFile.text);
    }
}

2 references
private void LoadCSVFile(string p_csvText)
{
    var csvRows = p_csvText.Split('\n');
    var csvRowHeaders = csvRows[0].Split(',');

    buildingDataParser = new BuildingDataParser(csvRowHeaders[0], csvRowHeaders[1], csvRowHeaders[2]);

    for (int item = 1; item < csvRows.Length; item++)
    {
        var csvRowItems = csvRows[item].Split(',');
        var Data = new BuildingDataParser.BuildingDataParser_Data();

        Data.index = int.Parse(csvRowItems[0]);
        Data.height = float.Parse(csvRowItems[1]);
        Data.intHeight = float.Parse(csvRowItems[2]);

        buildingDataParser.parser_Datas.Add(Data);
    }
}
}

[System.Serializable]
5 references
public class BuildingDataParser
{
    [System.Serializable]
    2 references
    public class BuildingDataParser_Header
    {
        public string indexName;
        public string HeightName;
        public string intHeightName;
    }

    [System.Serializable]
    3 references
    public class BuildingDataParser_Data
    {
```



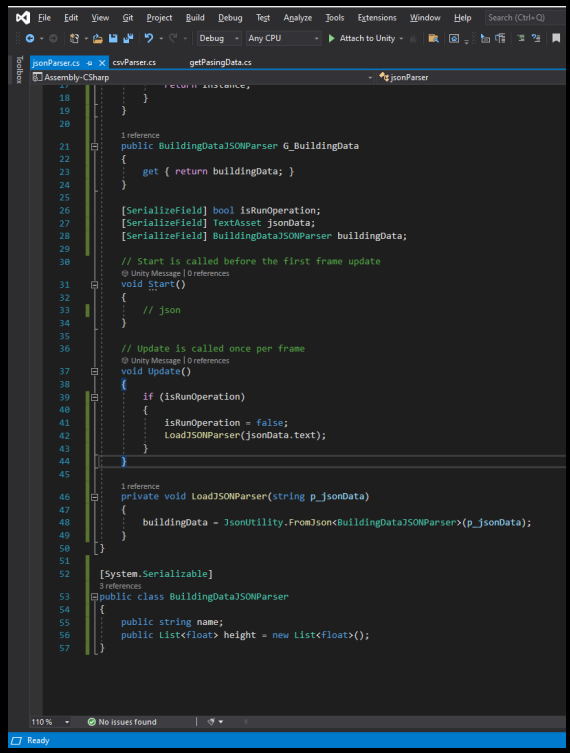
```
            var Data = new BuildingDataParser.BuildingDataParser_Data();

            Data.index = int.Parse(csvRowItems[0]);
            Data.height = float.Parse(csvRowItems[1]);
            Data.intHeight = float.Parse(csvRowItems[2]);

            buildingDataParser.parser_Datas.Add(Data);
        }
    }
}

[System.Serializable]
5 references
public class BuildingDataParser
{
    [System.Serializable]
    2 references
    public class BuildingDataParser_Header
    {
        public string indexName;
        public string HeightName;
        public string intHeightName;
    }

    [System.Serializable]
    3 references
    public class BuildingDataParser_Data
    {
        public int index;
        public float height;
        public float intHeight;
    }

    public BuildingDataParser_Header parser_Header;
    public List<BuildingDataParser_Data> parser_Datas = new List<BuildingDataParser_Data>();

    1 reference
    public BuildingDataParser(string p_indexName, string p_HeightName, string p_intHeight)
    {
        parser_Header = new BuildingDataParser_Header();
        parser_Header.indexName = p_indexName;
        parser_Header.HeightName = p_HeightName;
        parser_Header.intHeightName = p_intHeight;
    }
}
```

Sample Code for CSV Parser

Data Generated Dashboard | Read Exported Data into Unity

Create a JSON Parser

```using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[ExecuteInEditMode]
Unity Script (1 asset reference) | 4 references
public class jsonParser : MonoBehaviour
{
    private static jsonParser Instance = null;
    1 reference
    public static jsonParser GS_Instance
    {
        get
        {
            if (Instance == null)
            {
                Instance = FindObjectOfType<jsonParser>();
            }
            return Instance;
        }
    }

    1 reference
    public BuildingDataJSONParser G_BuildingData
    {
        get { return buildingData; }
    }

    [SerializeField] bool isRunOperation;
    [SerializeField] TextAsset jsonData;
    [SerializeField] BuildingDataJSONParser buildingData;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        // json
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (isRunOperation)
        {
            isRunOperation = false;
            LoadJSONParser(jsonData.text);
        }
    }
```



```    1 reference
    public BuildingDataJSONParser G_BuildingData
    {
        get { return buildingData; }
    }

    [SerializeField] bool isRunOperation;
    [SerializeField] TextAsset jsonData;
    [SerializeField] BuildingDataJSONParser buildingData;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        // json
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (isRunOperation)
        {
            isRunOperation = false;
            LoadJSONParser(jsonData.text);
        }
    }

    1 reference
    private void LoadJSONParser(string p_jsonData)
    {
        buildingData = JsonUtility.FromJson<BuildingDataJSONParser>(p_jsonData);
    }
}

[System.Serializable]
3 references
public class BuildingDataJSONParser
{
    public string name;
    public List<float> height = new List<float>();
}
```

Sample Code for JSON Parser

Create a Dashboard from Data

Creating a Visual Dashboard

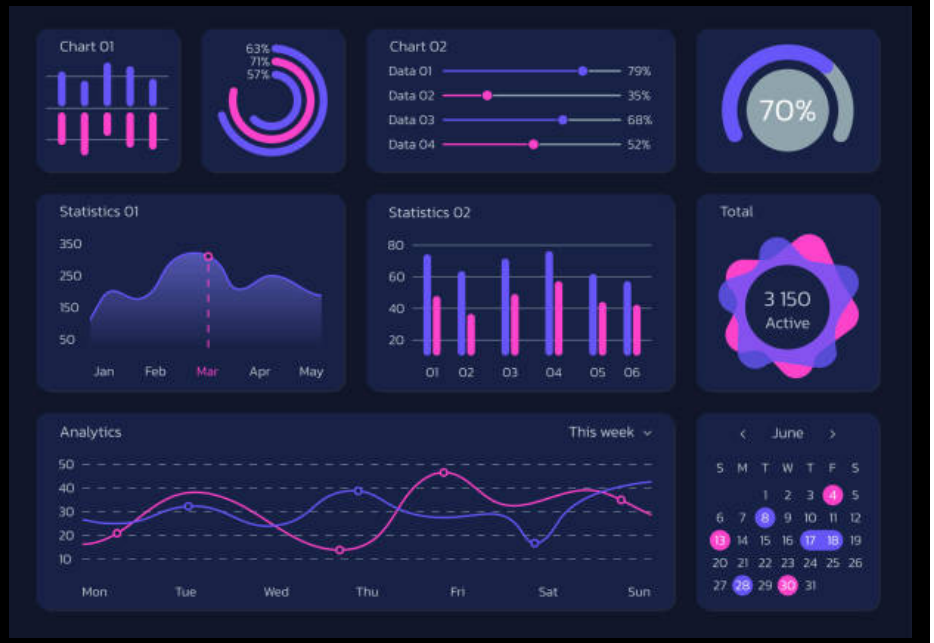
To achieve a visual dashboard you need to understand the data you have and the different possible ways to visualise it for better comphension though a dashboard.

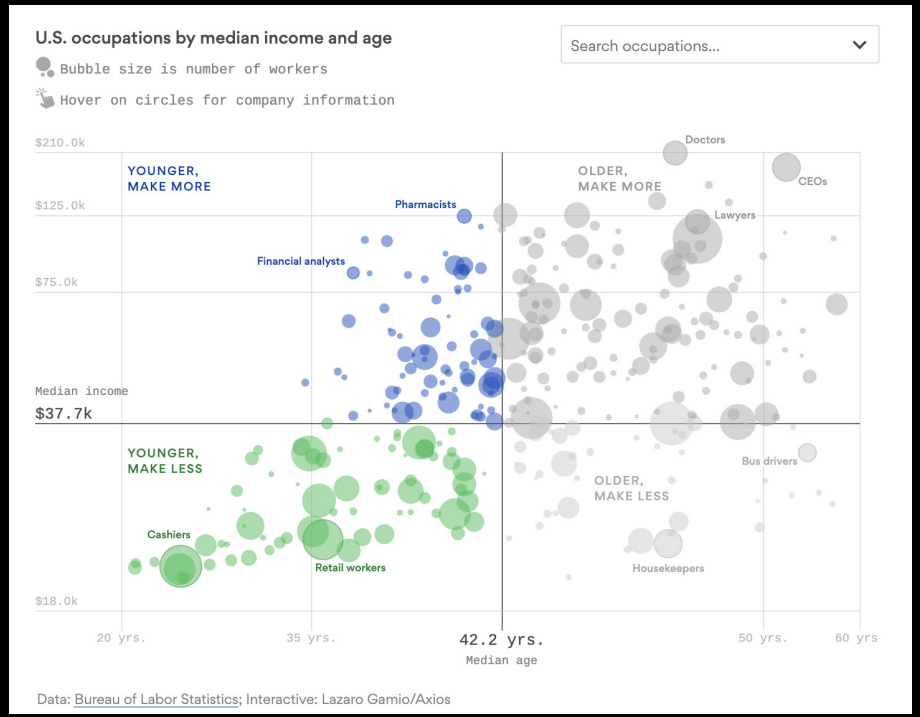
This would require looking at examples to see how similar data is represented, when look at the tools available to us to do so.

Creating a Visual Dashboard

Data Generated Dashboard | Create a Dashboard from Data

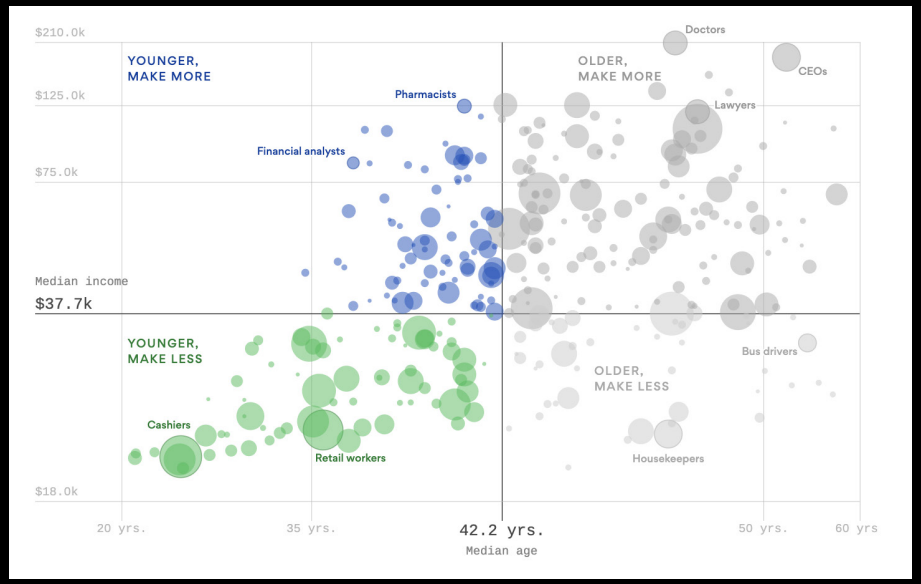
Creating a Visual Dashboard

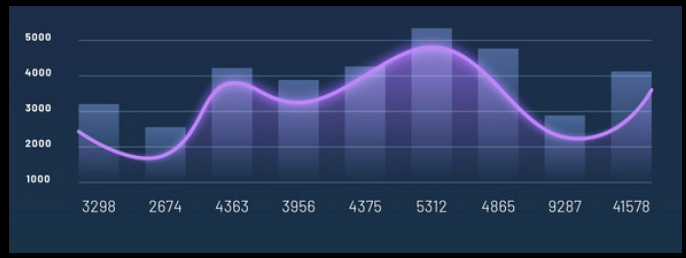




Data Generated Dashboard | Create a Dashboard from Data

Creating a Visual Dashboard





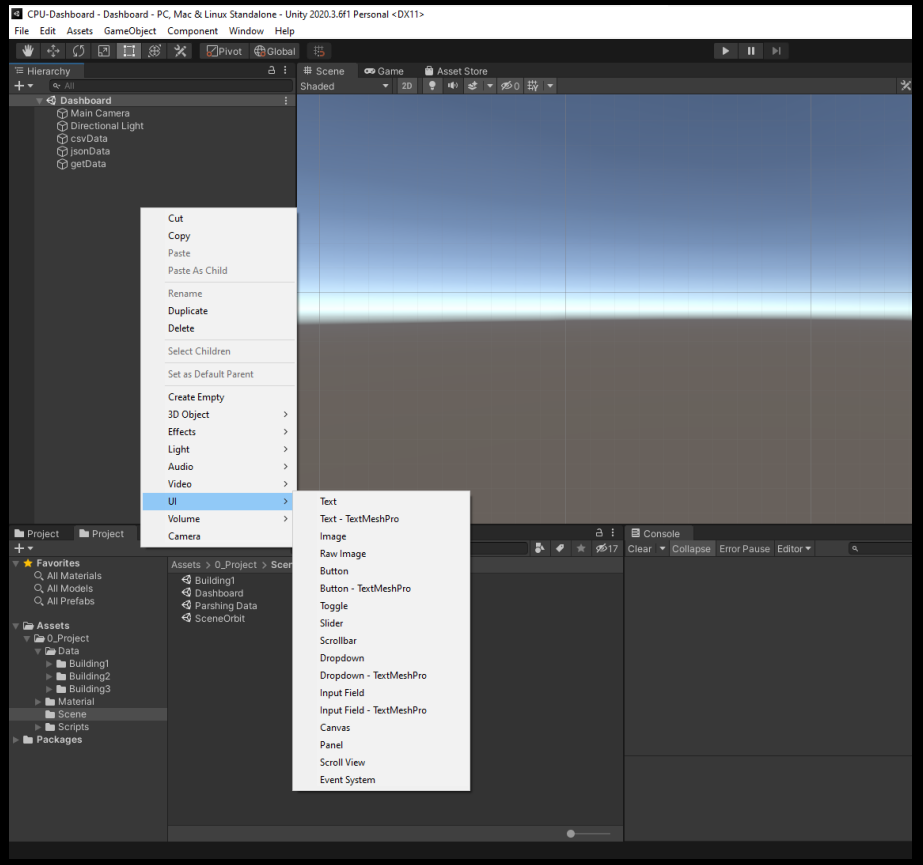
Chosen Dashboard to Create

Data Generated Dashboard | Create a Dashboard from Data

Creating a Visual Dashboard

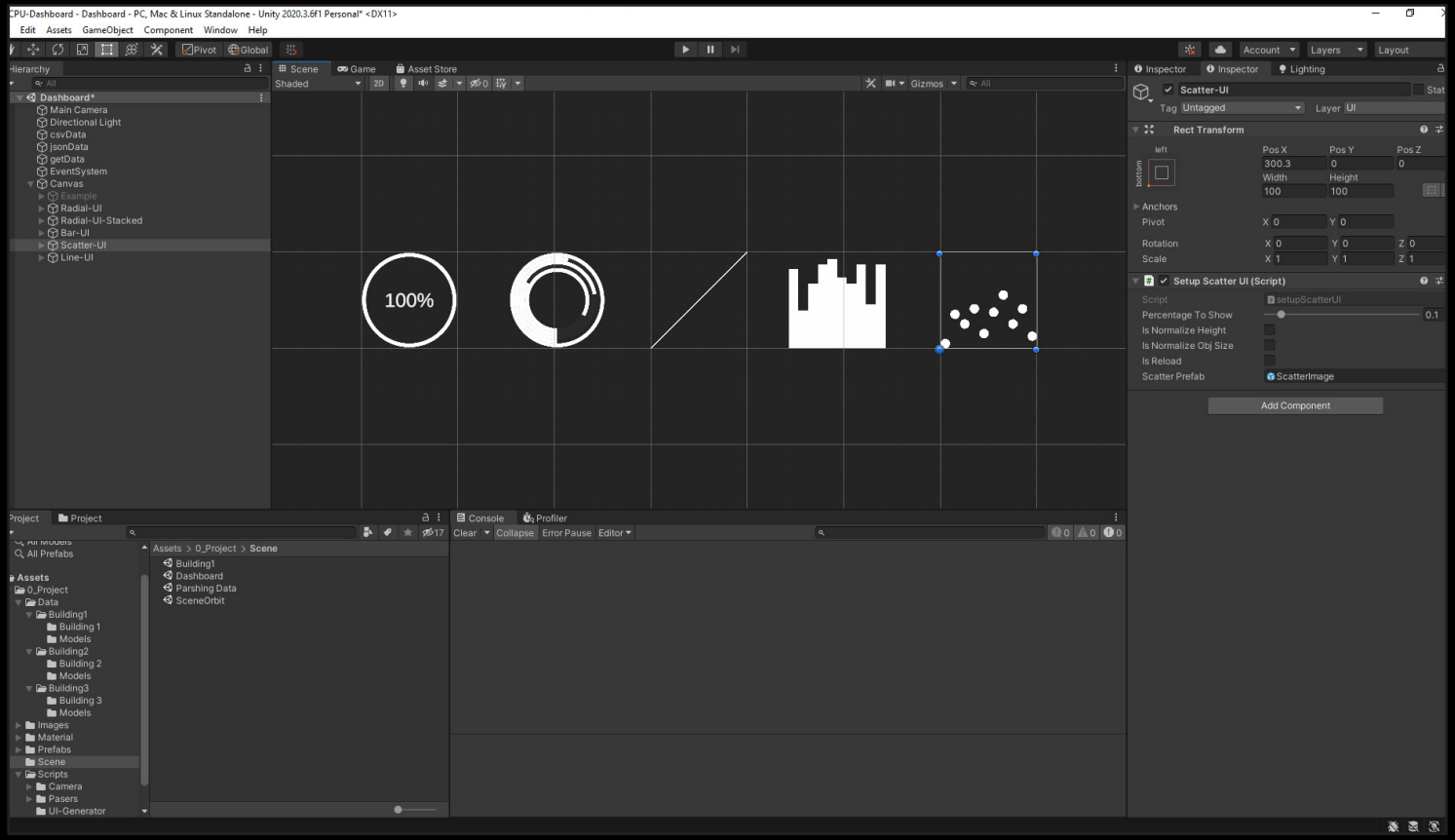
We will need to utilise the UI system inside of Unity which can be found in the Right click context menu over the Hierarchy and under **UI**. A canvas is required as a container for the UI elements

You can use many third party plugins to offset much of the setup but some of these come with a cost associated with them, which you can find on the asset store.



Demo

Creating a Visual Dashboard



Mock UI

Data Generated Dashboard | Create a Dashboard from Data

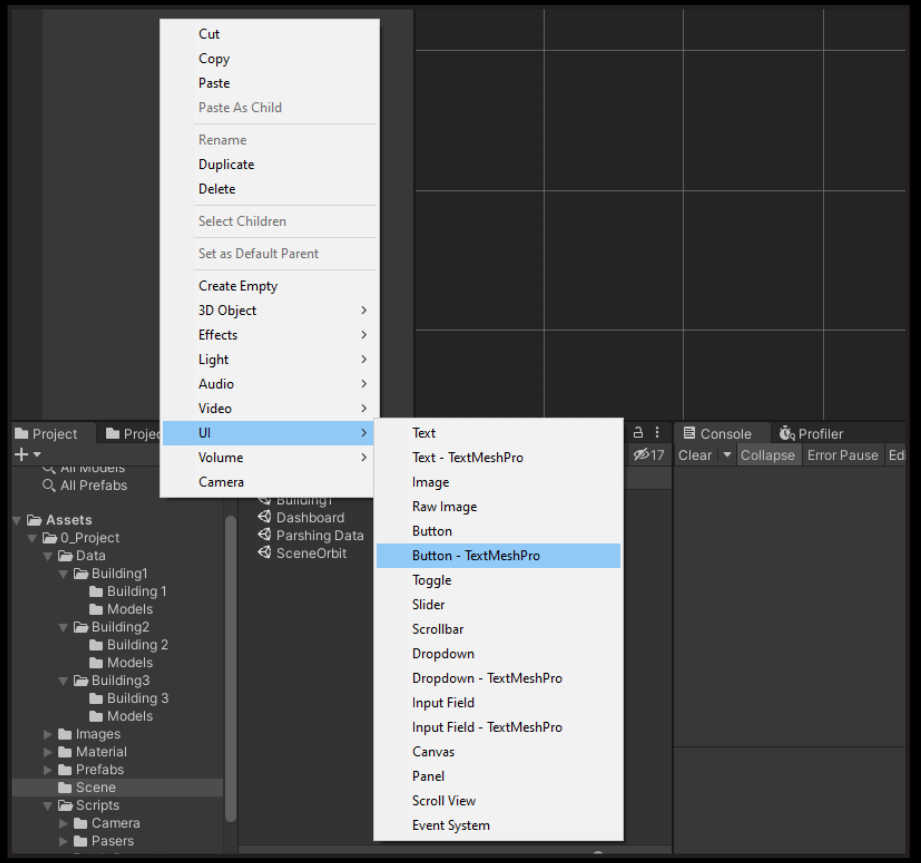
Add Basic Interaction

Basic Interaction

Unity offers a variety of tool to add interaction to our scene. This includes buttons, sliders, toggles and input fields. You can mix an add different things together to create custom interfaces.

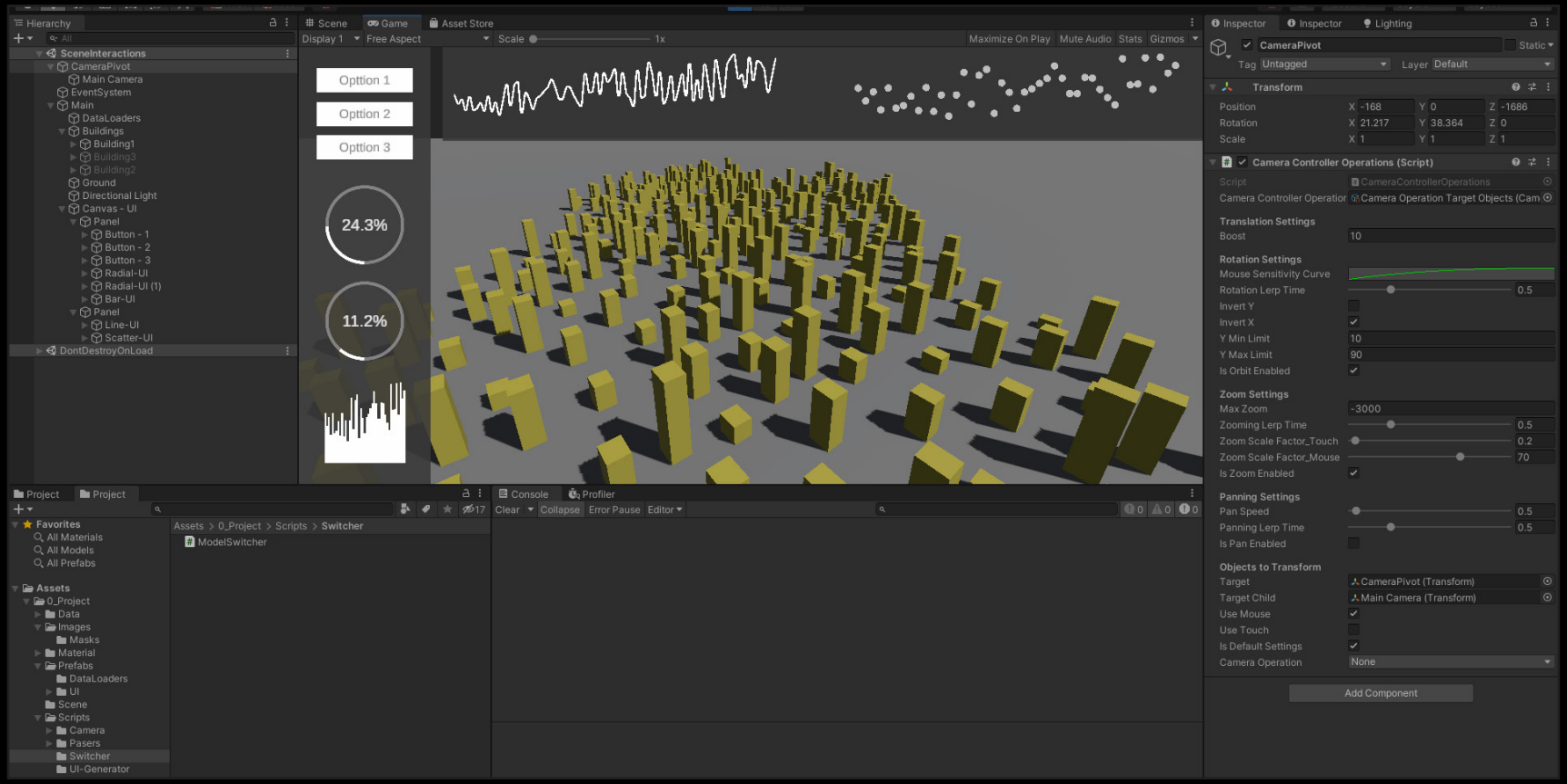
An Orbit Script will also be provided.

We will go through setting up the initial scene.



Demo

Add Basic Interaction

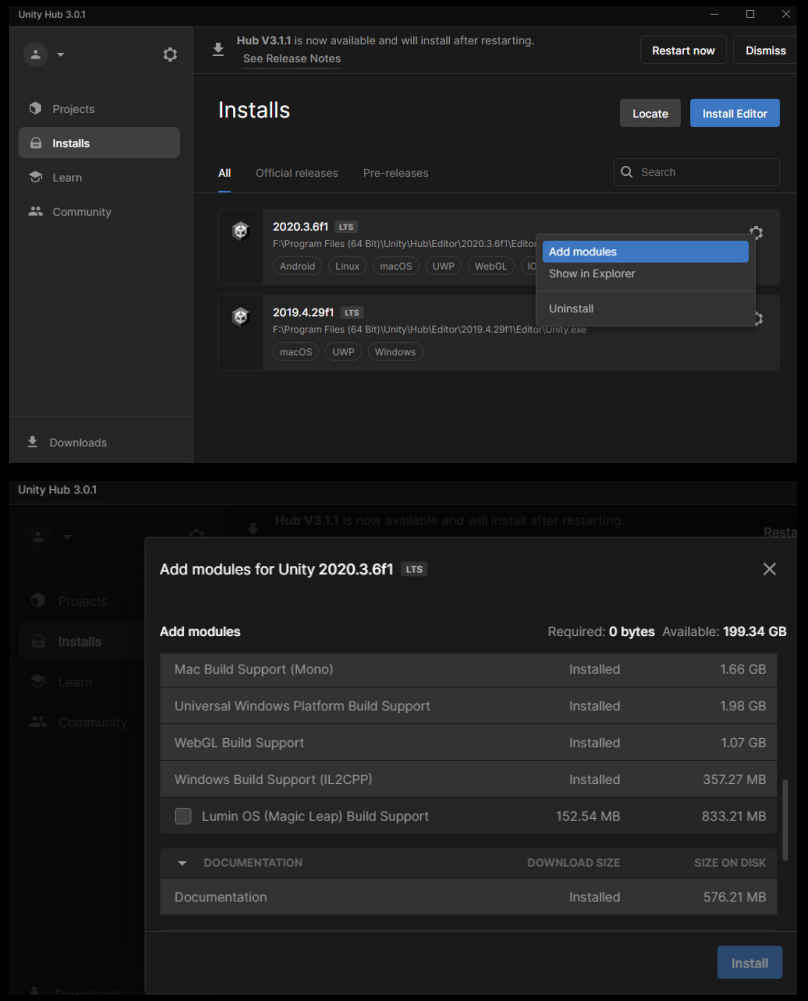


Export to WebGL

Export To WebGL

WebGL is a module that allows you to be able to export a web version of you project, which you can then later host.

It does need to be installed from the Add module section in side of Unity Hub. This does need to relate to the version of Unity you are used. Each different version will require its own installation.

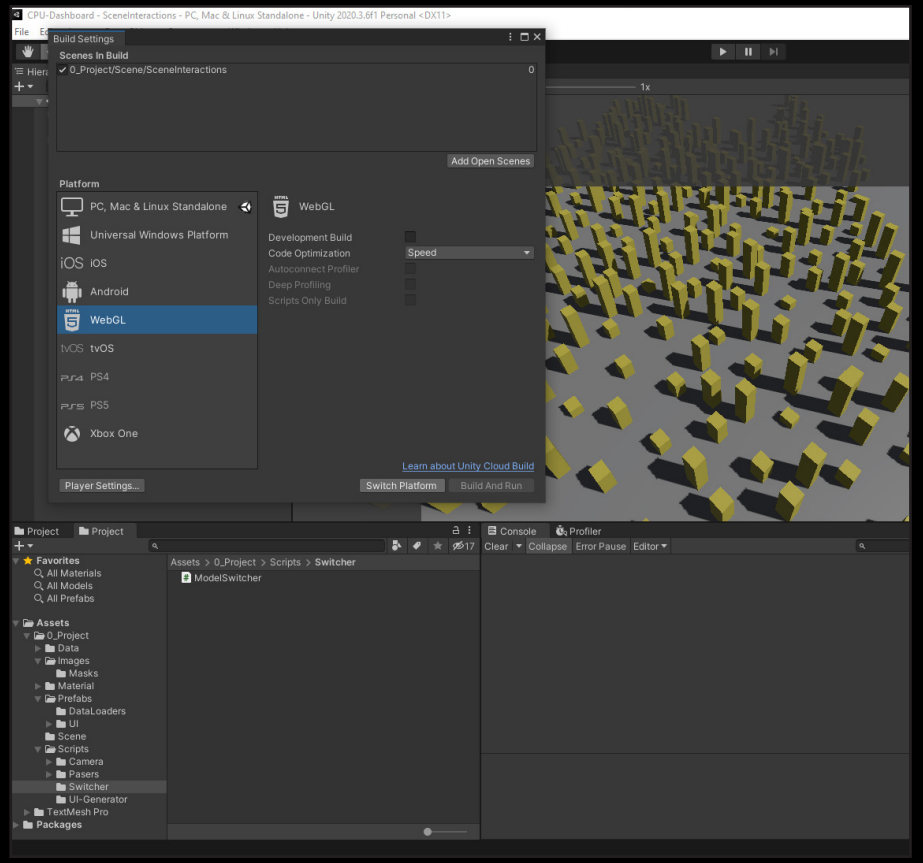


Exported Project Files

To Export the Project

Instruction:

+ Go to File > Build Settings
+ Add the Scene to the Scene in build Section
+ If You haven't switched to WEbGL, click WebGL under platform and click Switch Platform.
+ Now Click Build to Export.
+ If all goes well you will see build succeded when finished.



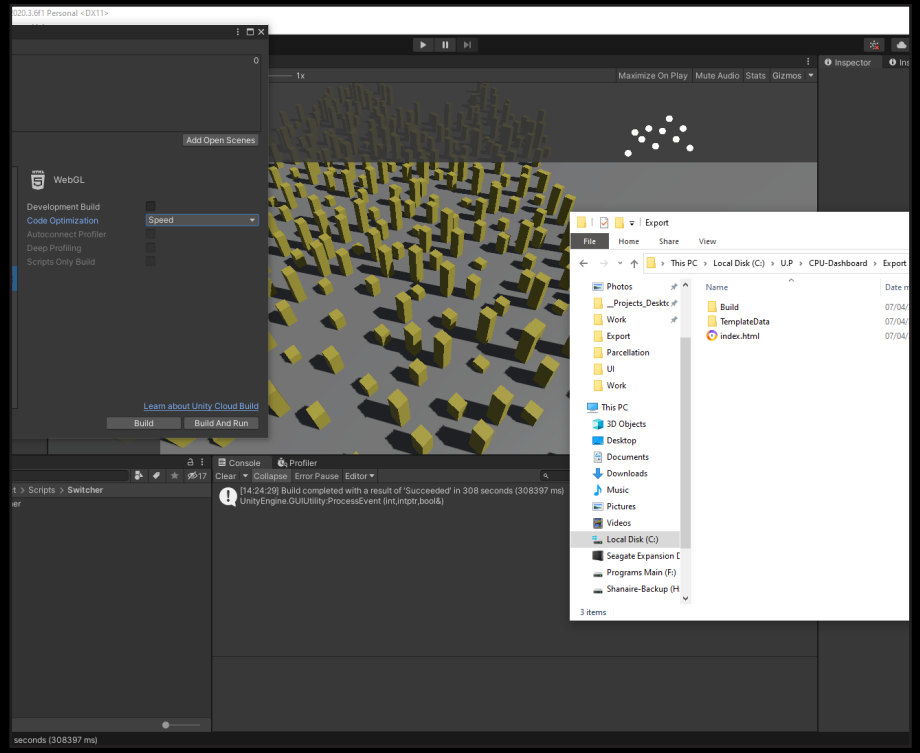
*If you have issues in running the export try changing Compression Format to disabled (File > Build Settings > Player Settings > Publishing Settings > Compression Format)

Export To WebGL

With the project now exported you should have files similar to that on the screen shot on the right.

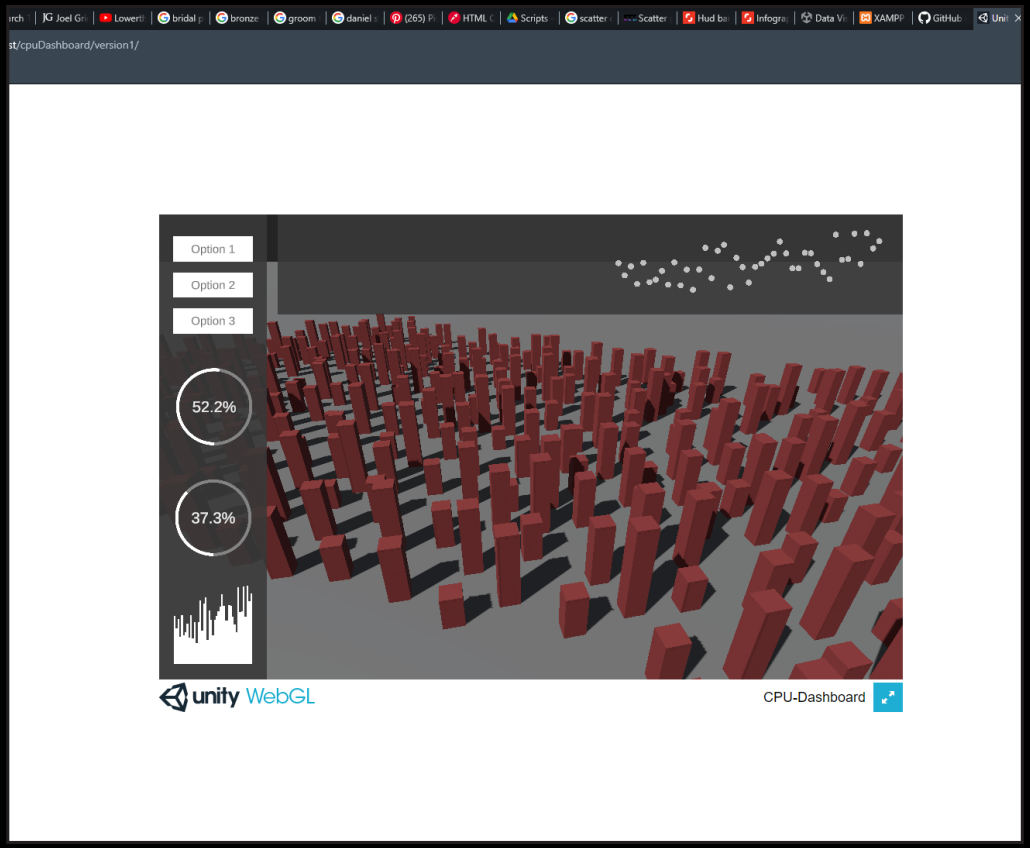
To view the files, now you need to either have a webserver running locally or over the internet to view the content.

You can use XAMPP to run a local server for testing purposes.



Export To WebGL

With the project now exported you should have files similar to that on the screen shot on the right.

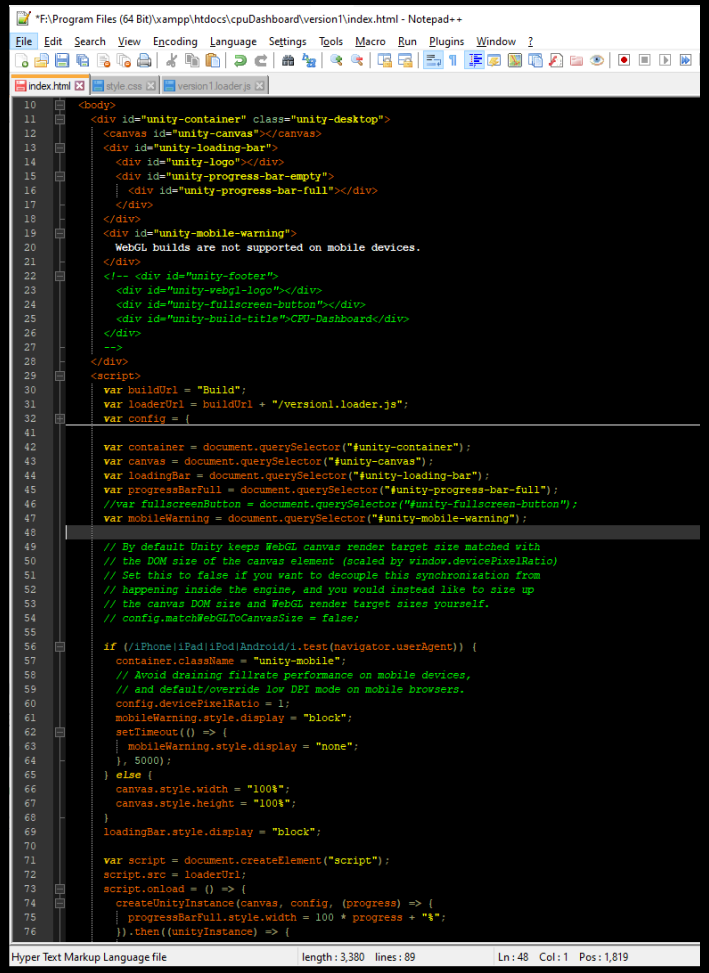


Remove Unity Footer & make fullscreen

To Remove the footer, comment out the all in the div tag from line 22 to 26 in the index.html file.

Also comment out the var fullscreenButton line on line 46 and 78 to 80. To prevent a null reference.

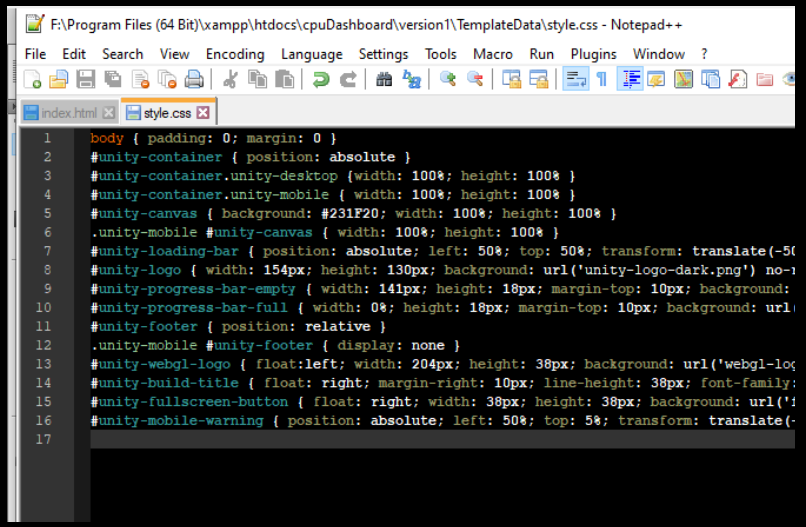
On line 12, remove the width and height next to the unity-canvas id and on line 66 and 67 change the values from a fixed one to 100% next to the unity-canvas id.



Remove Unity Footer & make fullscreen

Update the css file in the TemplateData folder to reflect the screenshot, as this will allow the application to now launch filling the screen.

Line 3 and Line 5, will need to be updated to 100% width and height.



* Once these settings have been changed, you will only need to update the build folder, you will need to make sure that you have a version in a different location to where you are building the actual project, so that you don't need to repeat the previous steps.

Points to Consider

Points to Consider

Check The Responsiveness of the application on different screen sizes
(Look at utilizing anchors on UI elements and aspect ratio filter script)

Utilising baking lighting to improve performance
rather than real-time lights

Optimising 3D models when importing into Unity
(This is related to polygon count, and less is better for good FPS)

Make sure scale is 1 to 1 from the native modelling package
(Ideal for calculating lighting)

Stay Curious!