Project

We are going to set up a data-importing system from Google Sheets to Unity that will let us switch between different scenes that read the imported data.

The project will contain all the completed scripts. Our task is to export the data correctly so the Unity game can run.

Sebastian Grygorczuk

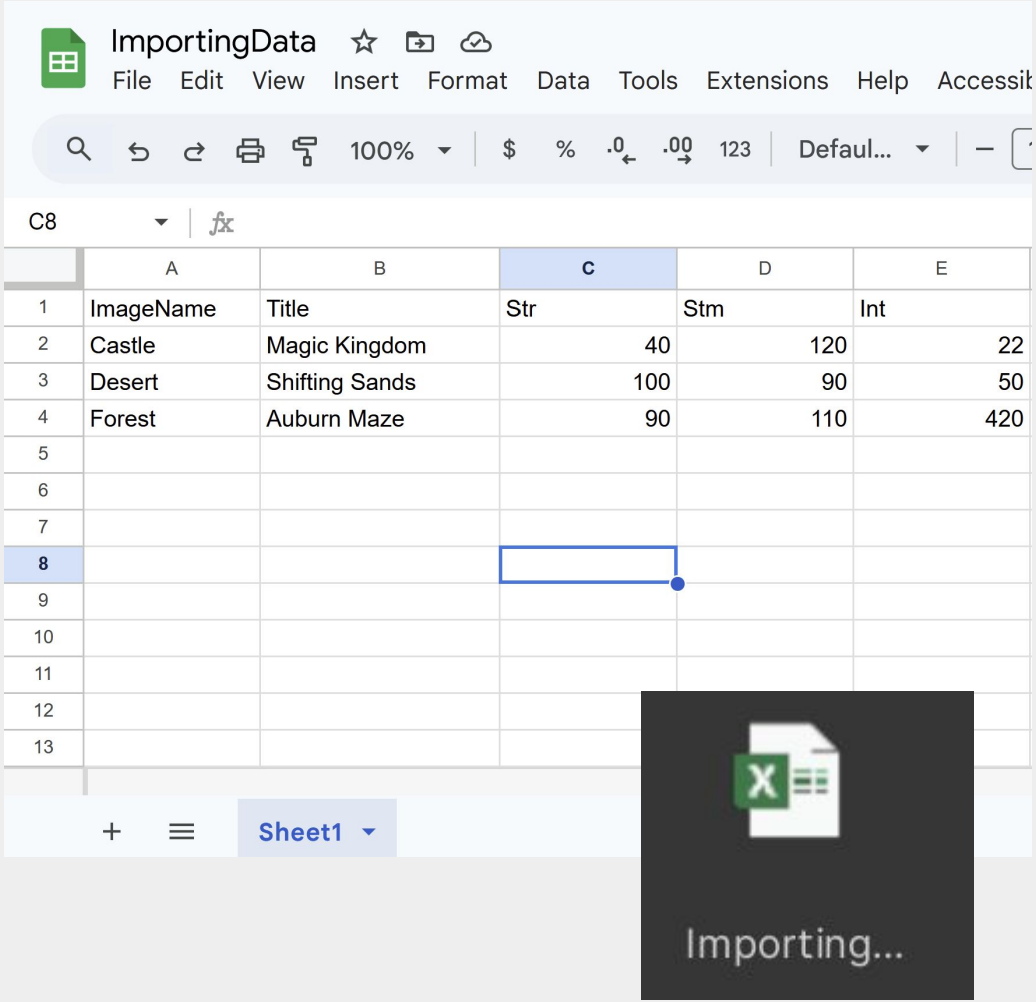
Google Sheets

First, we will use this Excel sheet, which has five columns and three rows.

The first row represents the different fields.

We will also use the active sheet named “Sheet1” to create the JSON file.

The ImportingData.xlsx file is in the Assets folder. Please upload it to your Google Drive so it can be modified.



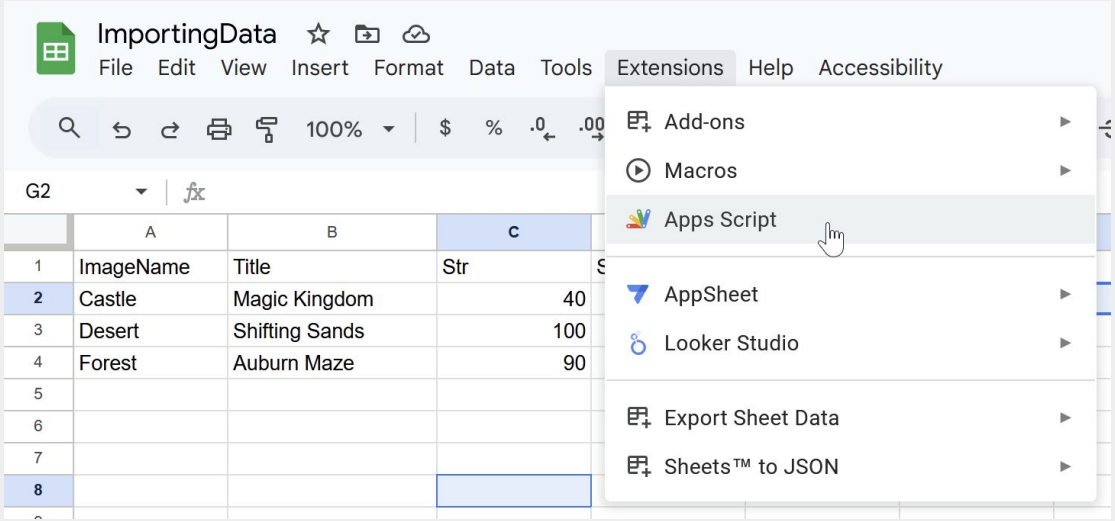
	A	B	C	D	E
1	ImageName	Title	Str	Stm	Int
2	Castle	Magic Kingdom	40	120	22
3	Desert	Shifting Sands	100	90	50
4	Forest	Auburn Maze	90	110	420

Sebastian Grygorczuk

Apps Scripts

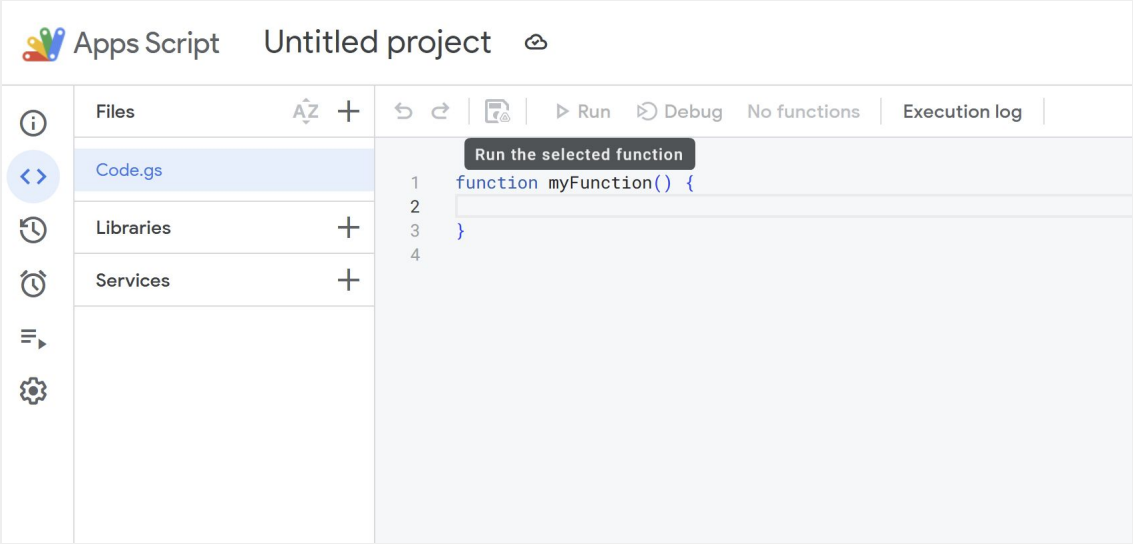
We will need to create our own script to read the data and organize it into JSON format.

To do that, go to **Extensions → Apps Script**.



Sebastian Grygorczuk

Apps Scripts



This window shows the script connected to the Google Sheet.

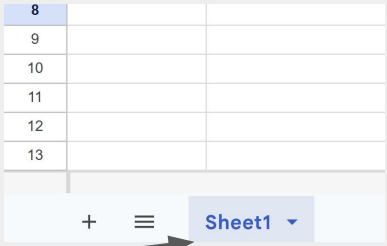
At the moment it is empty, but we will fill it with actions to convert the tables into JSON format that we can use.

exportJsonToDrive

1

```
function exportJsonToDrive() {
  // change the sheet name if needed
  const sh = SpreadsheetApp.getActive().getSheetByName('Sheet1');

  const values = sh.getDataRange().getValues();

  const headers = values.shift();
}
```

2



3

	A	B	C	D	E
1	ImageName	Title	Str	Stm	Int
2	Castle	Magic Kingdom	40	120	22
3	Desert	Shifting Sands	100	90	50
4	Forest	Auburn Maze	90	110	420

4

1	ImageName	Title	Str	Stm	Int

All the code for this is written in JavaScript.

1) The first function name will be changed to exportJsonToDrive so we know what it does.
2) Next, we connect to the active sheet with the specified name **"Sheet1."**
3) We get all the values from the rows and columns ([[row1col1, row1col2, ...], [row2col1, row2col2, ...]]).
4) We then save the first row as the headers, which are the variables we will collect for our game objects: ["ImageName", "Title", "Str", "Stm", "Int"].

Sebastian Grygorczuk

Connecting Headers to Values

```
function exportJsonToDrive() {
 ...
  // build array of row objects
  const rows = values
    .filter(r => r.join('') !== '')
    .map(r => Object.fromEntries(headers.map((h,i) => [h, r[i] === '' ? null : r[i]])));
}
```

Using the headers, we go through each row and create JSON sections. Example:

[{ "ImageName": "Castle", "Title": "Magic Kingdom", "Str": 40,"Stm": 120, "Int": 22 },

...

{ "ImageName": "Forest", "Title": "Auburn Maze", "Str": 90, "Stm": 110, "Int": 420 }]

Adding a Name to the JSON Rows

```
function exportJsonToDrive() {
     ...
  const payload = { levelInfo: rows };
}
```

We add a top-level name to the JSON so we can later match it to the data structure in Unity with the same name.

[**“levelInfo”:** { "ImageName": "Castle", "Title": "Magic Kingdom", "Str": 40,"Stm": 120, "Int": 22 },

...

{ "ImageName": "Forest", "Title": "Auburn Maze", "Str": 90, "Stm": 110, "Int": 420 }]

Formating to look like JSON

```
const json = JSON.stringify(payload, null, 2);
```

1. This step takes the rows and converts them into JSON formatting.
2. null means include all values found in the cells.
3. 2 means indent with two spaces for readability.

```
const name = data.json';
```

Sets up a string with the name of the file where you want to save the JSON data.

```
{
    "levelInfo":[
      {
        "ImageName": "Castle",
        "Title": "Magic Kingdom",
        "Str": 40,
        "Stm": 120,
        "Int": 22
      },
      {
        "ImageName": "Desert",
        "Title": "Shifting Sands",
        "Str": 100,
        "Stm": 90,
        "Int": 50
      },
  ]
}
```

Creating the File

```
// overwrite if file exists, else create new
const files = DriveApp.getFilesByName(name);
if (files.hasNext()) {
  files.next().setContent(json);
} else {
  DriveApp.createFile(name, json, MimeType.PLAIN_TEXT);
}
```

Checks if the file already exists in your Google Drive.

If it exists, overwrite the existing file.

If it does not exist, create a new file with the given name.

Full Code

```
function exportJsonToDrive() {
  // change the sheet name if needed
  const sh = SpreadsheetApp.getActive().getSheetByName('Sheet1');
  const values = sh.getDataRange().getValues();
  const headers = values.shift();

  // build array of row objects
  const rows = values
    .filter(r => r.join('') !== '')
    .map(r => Object.fromEntries(headers.map((h,i) => [h, r[i] === '' ? null : r[i]])));

  const payload = { levelInfo: rows };
  const json = JSON.stringify(payload, null, 2);
  const name = 'data.json';

  // overwrite if file exists, else create new
  const files = DriveApp.getFilesByName(name);
  if (files.hasNext()) {
    files.next().setContent(json);
  } else {
    DriveApp.createFile(name, json, MimeType.PLAIN_TEXT);
  }
}
```

Sebastian Grygorczuk

Run the Script

Apps Script Json Unsaved changes

Files

Code.gs

Libraries

Services

Run Debug exportJsonToDrive Execution log

```
function exportJsonToDrive() {
  // change the sheet name if needed
  const sh = SpreadsheetApp.getActive().getSheetByName('Sheet1');
  const values = sh.getDataRange().getValues();
  const headers = values.shift();

  // build array of row objects
  const rows = values
    .filter(r => r.join('') !== '')
    .map(r => Object.fromEntries(headers.map((h,i) => [h, r[i] === '' ? null : r[i]])));

  const payload = { levelInfo: rows };
  const json = JSON.stringify(payload, null, 2);
  const name = 'data.json';

  // overwrite if file exists, else create new
  const files = DriveApp.getFilesByName(name);
  if (files.hasNext()) {
    files.next().setContent(json);
  } else {
    DriveApp.createFile(name, json, MimeType.PLAIN_TEXT);
  }
}
```

Click **Run** to execute the script. You will receive several pop-ups asking for permission to connect the script to your account. Approve them to continue.

Sebastian Grygorczuk

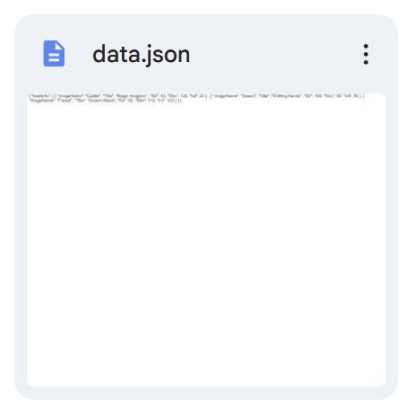
The File

Run Debug exportJsonToDrive Execution log

```
function exportJsonToDrive() {
  // change the sheet name if needed
  const sh = SpreadsheetApp.getActive().getSheetByName('Sheet1');
  const values = sh.getDataRange().getValues();
  const headers = values.shift();

  // build array of row objects
  const rows = values
    .filter(r => r.join('') !== '')
    .map(r => Object.fromEntries(headers.map((h,i) => [h, r[i] === '' ? null : r[i]])));

  const payload = { levelInfo: rows };
  const json = JSON.stringify(payload, null, 2);
  const name = 'data.json';

  // overwrite if file exists, else create new
```

Execution log

3:07:50 PM	Notice	Execution started
3:07:52 PM	Notice	Execution completed



Once you run the script, the Execution Log (console) will indicate whether the job was completed successfully.

Since we did not specify a folder, the data.json file will be created in the main folder of your Google Drive account.

Imported Object

Data (Text Asset)

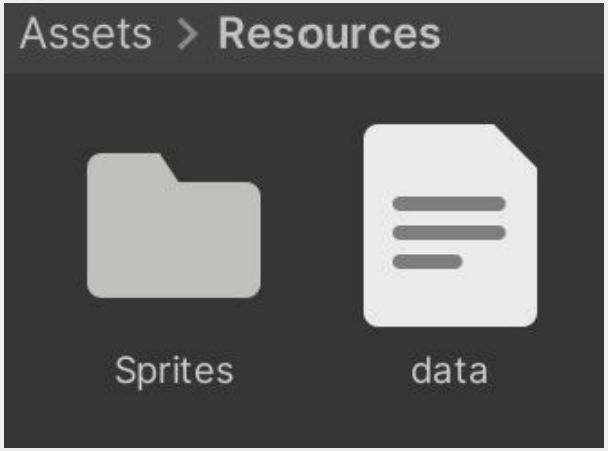

```
{
  "levelInfo":[
    {
      "ImageName": "Castle",
      "Title": "Magic Kingdom",
      "Str": 40,
      "Stm": 120,
      "Int": 22

    },
    {
      "ImageName": "Desert",
      "Title": "Shifting Sands",
      "Str": 100,
      "Stm": 90,
      "Int": 50

    },
    {
      "ImageName": "Forest",
      "Title": "Auburn Maze",
      "Str": 90,
      "Stm": 110,
      "Int": 420

    }
]
```


Into Unity

We are going to download that file and place it in the **Assets** folder.

To access the data at runtime, we need to put it in a special folder that Unity can read during gameplay. That folder is **Resources**.

In this example, we are also connecting one of the fields to an image.



Unity Scripts

We will have three scripts that we need:

1) **LevelInfoData** – structures the data so we can read the JSON fields correctly.
2) **DataBase** – reads the data from the JSON file.
3) **ViewController** – lets us change things in the game using the data we imported.

LevelInfoData

A general structure contains all the same fields as a JSON row.

We copy each row into one of these structures.

```
[Serializable]
public class LevelInfoData
{
    public string ImageName;
    public string Title;
    public int Str;
    public int Stm;
    public int Int;
}
```

LevelInfoDatabase

```
[Serializable]
public class LevelInfoDatabase {
    public LevelInfoData[] levelInfo;
}
```

This is where we load all of the rows exactly as they appear in the JSON file.

LevelInfo

This is a special class that our **ViewController** will use.

It changes the **ImageName** field from a string to a **Sprite** so we can swap the pictures.

It also has a constructor so we can create new objects with the given values.

```
[Serializable]
public class LevelInfo
{
    public Sprite Sprite;
    public string Title;
    public int Str;
    public int Stm;
    public int Int;

    public LevelInfo(LevelInfoData data, Sprite sprite)
    {
        this.Sprite = sprite;
        this.Title = data.Title;
        this.Str = data.Str;
        this.Stm = data.Stm;
        this.Int = data.Int;
    }
}
```

Levels List

DataBase is the generic data game object that will read the JSON file.

All of our data will be stored in **levels**, which is a list of the fully realized objects that the controller can connect to.

```
public class DataBase : MonoBehaviour {

   public List<LevelInfo> levels = new List<LevelInfo>();

}
```

Reading the JSON

The first step is to look in the **Resources** folder for a text file named data.json (the .json extension is not required).

Next, we use **FromJson**, which will automatically fill the **LevelInfoDatabase** array with all the rows we created.

```
public class DataBase : MonoBehaviour {

    public List<LevelInfo> levels = new List<LevelInfo>();

    public void ReadData() {
        // Gets the Json from Resource Folder
        TextAsset jsonFile = Resources.Load<TextAsset>("data"); // no .json extension
        // Reads the JSON
        LevelInfoDatabase levelInfoDatabase = JsonUtility.FromJson<LevelInfoDatabase>(jsonFile.text);
    }
}
```

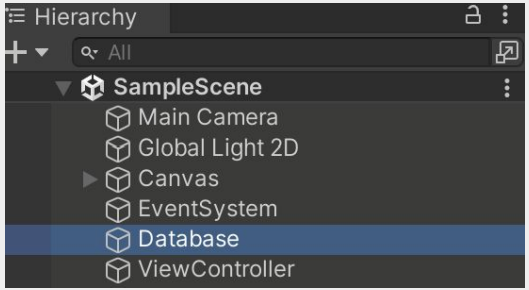
Translating the Data

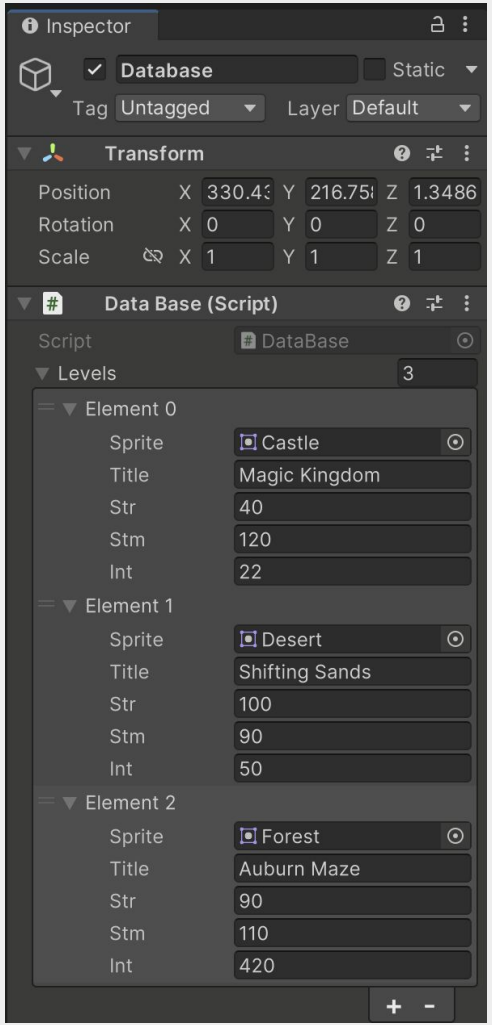
We could stop at that last point if our data did not need to be modified, but instead we will go through all the entries in the **DataBase** we imported.

For each entry, we create a new object that copies the **Title**, **Str**, **Stm**, and **Int**, but converts the **ImageName** into a **Sprite** that also exists in the **Resources** folder.

```
public class DataBase : MonoBehaviour {
   public List<LevelInfo> levels = new List<LevelInfo>();

   public void ReadData() {
       ...
       // Uses the broken JSON data to collect the text and int values as well as connect to the
       // appropriate image in Resource Folder
       foreach (var t in levelInfoDatabase.levelInfo) {
           levels.Add(new LevelInfo(t,
               Resources.Load<Sprite>("Sprites/" + t.ImageName)));
       }
   }
}
```

Game Objects





Once all of that is set up, we have a game object called **Database** with the **DataBase** script attached to it.

When the game starts, the **ViewController** calls the **DataBase**, runs the **Read** function, and our **levels** list is populated with the imported data.