# Tips for developing in Unity

1. When importing images and videos, take a look at the import settings of each item.
2. Write your game in C#. JavaScript might seem easy to begin with, but ultimately it will cause massive headaches. The auto completion functions do not work properly for JavaScript and it’s not nearly as strict as C#. Do yourself and future developers a favor and go for C# within Unity.
3. Also, when you choose to develop with JavaScript, notice that JavaScript files are compiled BEFORE C# files. When integrating my files, note that the C# files need to be in the map called “Standard Assets”, otherwise you won’t be able to link them in your JS file.
4. Develop the game, make sure there are no bugs. Since Unity isn’t the most stable engine ever, even the most minor bug may break the whole game and/or framework it will be running on.
5. When you import assets, make sure you import just the ones you need. Do not import the whole Standard Assets map. Importing everything will make your project way too big and will take longer while loading.
6. Last but not least, actually a very important note… Make sure you use good names for everything. I’ve seen a lot of projects and games with weird names and it will be really hard to find back certain stuff when your project gets bigger. So give your variables and other stuff obvious names.

# Important note

*When I’m talking about ‘gamename’ it always needs to be the same and in lowercase. It’s the name of your .exe file, the name in the database and the name in front of images. For example: safety.exe has safety\_uitleg\_2.jpg as name of instruction screen.*

# Connecting your game to the mainframe

So, you’re game is all done and you want to connect it with the system. It’s pretty easy don’t worry, but you’ll need to do some extra coding. Well, let’s start.

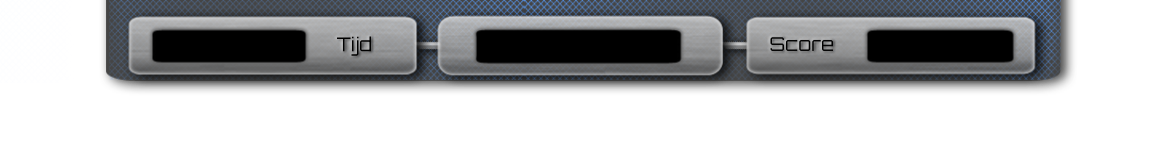
There are 2 files needed to get a connection with the whole system. I will explain, with just a few words, what both files do.

* *Arguments.cs gets the arguments out of the commandline which was send by the mainframe system. The following arguments are available: userID, gameID, username, gametime.*
* *DBconnection.cs has a function that uploads the score of your game.*

1. Import Arguments.cs and DBconnection.cs into your project and into your scene.
2. Once you’ve linked Arguments.cs you can get the username of the player by *<argumentslink>.getUsername().* Also, use *.getGameTime()* when your game needs to have a timelimit. It gets the value from the database.
3. To upload the score of the game when a player’s finished, use the following *StartCoroutine(<dbconlink>.UploadScore(<argumentslink>.getUserID(), <scoreofplayer>)).* **Make sure this gets called only once! Otherwise you will overload the database.**
4. Every game needs 2 explanation screens, these need to be .jpg files. To import these files you will need to rename your images to <gamename>\_uitleg\_1.jpg and <gamename>\_uitleg\_2.jpg. Put these images in the following map on every console: C:\mainframe\assets\intro.

# Implementing the top HUD

As you’ve probably seen, most games in the Radar & Detectie room has a top HUD.



If you want or need to implement this HUD, just follow these easy steps.

1. Create a map called ‘Resources’ in your assets folder and import the ‘HUD’ folder included with this document.
2. Go to the script where you need to implement this HUD and copy paste the code found in the document HUDAnimationCode.cs *(This is only for C# projects, when scripted in js you will need to write it yourself.)*

Well, that should be it. Good job ☺

# Assets

You will need to create a few more things to make sure everything will work as it should, so let’s get started. Examples are located in the ‘assets’ folder.

**Every game needs…**

1. …an idlescreen video, see assets/idle/veiligheid\_idlescreen\_movie. Ask het HEIM what they want the idlescreen to look like or come up with something nice, just make sure the name of the game stands out so everyone can see what game you can play there. The format should be 1920x1080, .OGV (Miro converter is pretty good for .OGV)
2. … introduction/instruction screens. These should consist out of 2 images. Take a look at the examples, see assets/intro/veiligheid\_uitleg\_\* . These are 1920 x 1080, JPG.

Important: You’ll see that the blue line does not continue in the lower right corner, that’s because the Next and Play button will be implemented there by the mainframe system.

1. … score items. Look at assets/score/veiligheid\_\*, you’ll see a badge and middle image. These are loaded by the mainframe to show the player’s score. Create 2 images with the same dimensions and characteristics and you’ll should be good.