Technical Documentation

Capstone 2022

Team Aistikattila

Eva Zorman, eva.h.zorman@utu.fi
Heidi Laine, heidi.k.laine@utu.fi
Jasperi Sivenius, jasperi.m.sivenius@utu.fi
Joona Wiik, joona.e.wiik@utu.fi
Xuexian Chen, xuchen@utu.f

March, 2023

Table of Contents

Introduction **2**

System Overview **2**

Basic Design and context 2

Decisions 3

Out-of-scope 3

Architectural design **4**

Unity program architecture 4

Program interfacing with the lights 4

Technical constraints encountered **4**

Blending issues between the projectors and Unity program 5

Scaling issues with the UI in the Aistikattila room 5

Providing realistic environment effects 5

Running the Python script via Unity 5

Introduction

This document describes some of the architectural decisions made during the design process of the Aistikattila Experience, as well as technical constraints encountered during this process and the workarounds decided upon.

System Overview

Basic Design and context

The Aistikattila Experience is designed to be a user-friendly and efficient software that would bundle the experience of setting up the virtual environment of the Aistikattila in Flavoria, which comes with a large overhead of necessary IT knowledge and time wasted, to ensure that everything is set up properly. As the virtual environment includes not only software aspects (such as running software on the computer in the room, the projectors connected to

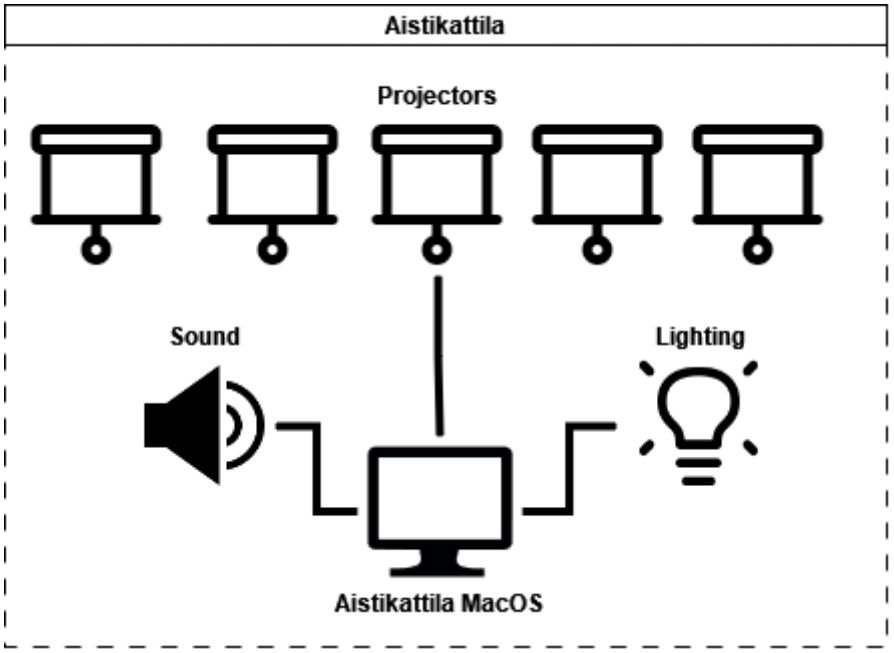
it, the sound equipment, etc.) but also certain hardware aspects (i.e. the lighting and scent machines available within the Aistikattila room), some interfacing between the different systems was necessary. As such, the team designed the software with the following points in mind:

- Ease of use
- Modularity
- Increased virtual experience

Ease of use is self-explanatory, and it was perhaps the most desired functionality of the project, as the end product aims at researchers, and not IT experts. With modularity, the team decided to design the software in ways to ensure that additional expansions to it are as simple as possible, and that additional plug-in functionality is possible, no matter if it had been thought of during the design process or not. Finally, the team wished to increase the virtual experience of research subjects in the Aistikatilla, which previously only provided still imagery audio-video experience.

Decisions

In order to support the aforementioned design wishes, the Aistikattila Experience was designed as a single Unity program with high modularity, which will then additionally interface with the lighting of the room. The decision for this was made to try and ensure that usage of the program can be expanded and used for a long time to come.



Out-of-scope

As the design and implementation of the Aistikattila Experience, a Capstone project which lasted approximately 6 months with irregular intervals of workloads available, the team has decided that certain aspects of the project are out-of-scope. The list of such things can be found below.

- Connecting the whole entirety of Aistikattila with the Aistikattila Experience software, as a home assistant had been introduced into the mix later down the line
- Connecting the scent machines with the Aistikattila Experience, as they proved to be quite difficult to interface with
- Creating an interactive scenery with the help of a Xbox Kinect and additional external software, due to the difficulty of interfacing a Xbox Kinect with a MacOS
- Allowing for users to create custom settings for sceneries they had additionally added to the program, as it would be incredibly difficult to provide that modularity ahead of time with unknown constraints (note that adding custom sceneries *is* supported)

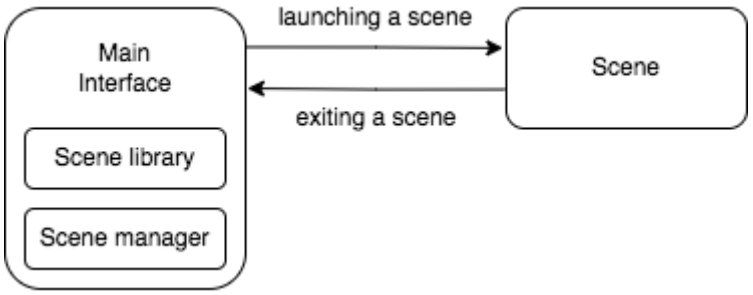
Architectural design

The state of the environment is completely controlled by the Unity program built from the source code of the Aistikattila Experience. This is due to the fact the team wished to reduce any unnecessary work being done in the background, to reduce spent resources, and avoid evoking system daemons that might at some point in the future clash with other events on the system.

Unity program architecture

The Aistikattila Experience is built as a Unity project, using Unity Editor version 2021.3.11f1. The program uses Unity's scenes functionality, with the main interface being the first scene in build, and the scenes played on the projectors being the next scenes in build.

In the main interface scene, it is possible to navigate between two views, the scene library, and the scene manager. Launching the scenes to the projectors and adjusting the settings happens from the scene library view. Viewing and deleting items from the scene library happens in the scene manager and adding a new scene is possible from the scene manager.

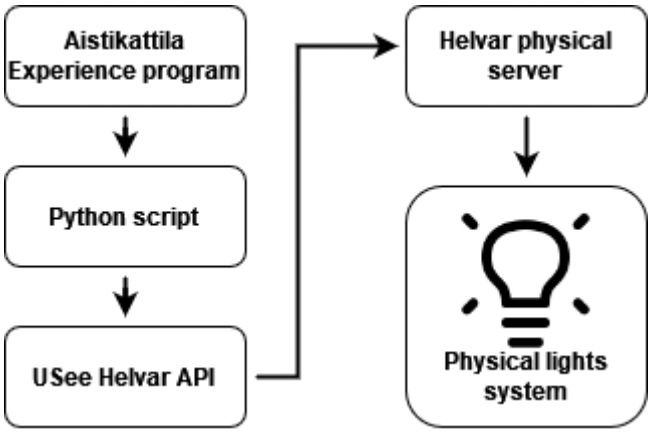


When a scene is being shown, the Unity program uses five displays. The first display is the iMac in Aistikattila and it states which scene is being shown and provides the option to exit from the scene. The displays 2-5 are the four projectors on the walls in the Aistikattila. When the scene is launched, the sound and lights are adjusted simultaneously.

When launching the Forest scenery, the program has three parameters that are being sent to the scene. These parameters are binary options Day/Night, Rainy weather/Sunny weather and Sound/No sound. The user has the option to toggle between these settings before launching the scene.

Program interfacing with the lights

As with all other aspects of the Aistikattila Experience, Unity handles evoking the script that interfaces with the lighting in the Aistikattila. Using a Python plugin for Unity, it has been possible to call a python script that in turn interfaces with the USee interface of the Helvar server available in the Aistikattila. The interfacing process can be seen in the image below.



A standalone Python script was designed to ensure that future interfacing with the lighting can be done by simply re-using the script by other users. The script uses the requests library

to evoke HTTP requests to the USee Helvar API provided by the Helvar company, as opposed to directly talking to the server. This is due to the fact that the interface has a defined REST API which can be simply called if the proper URIs are known. With this, we are using an already implemented solution, and manage to avoid having two working solutions instead.

Technical constraints encountered

While working on the Aistikattila Experience project, as with any other, certain technical constraints were encountered, which needed to be solved to the best of the team's capabilities. In most cases, they were tackled and successfully overcome.

Blending issues between the projectors and Unity program

The Aistikattila Experience is situated in a room with 12 metre walls with four slightly overlapping projectors. Our aim was to stitch the four overlapping projectors together to create a single screen as seamlessly as possible out of them. The solution built to create this single digital installation wall with multiple projectors proved to become a larger technical difficulty than originally planned.

Unity provides the inbuilt features for multiple cameras within a single scene, and multiple separate displays being connected at the same time to display the camera views. These features were both utilised in our solution. After having these basic structures to build on, the main aim of the solution was to work both as an edge blender between the projectors, and a keystone correctional tool to individualise the projection images to fit the physical requirements of the Aistikattila space. Firstly, the edges of adjacent overlapping projector screens are blended to both not seem more bright than the rest of the screens, but also to make the displayed image blend as effortlessly as possible to the one right next to it. Secondly, keystone correction (i.e. the capability for moving the corners of each projection for perspective distortion) was also a necessity for the finished product to look in any way professional, as the projectors aren't perfectly horizontally aligned within the room, and the physical structures (e.g. ceiling beams) within the room required aligning the projections in a specific manner.

After we were able to successfully enable both of these actions, a personalised JSON calibration file for the Aistikattila room was created for later usage. This calibration file can simply be used from now on in all sceneries used through the Aistikattila Experience Unity

application in connection with the projector warp system to make the process more effortless to the user with no need for readjusting each of the blending parameters for each scenery.

Scaling issues with the UI in the Aistikattila room

There were unexpected issues regarding the scaling of elements in the main interface scene, especially with the scene manager view. The list of custom-made scenes did not appear correctly on the Aistikattila iMac, even though there were no problems on any other computer, either Macbook or PC. Our team managed to reach a custom-made solution by testing and implementing different values for element positioning, but because of time constraints, we were unable to find the root cause for this issue.

Providing realistic environment effects

When it comes to creating realistic environmental effects in this Unity project, there are several key elements that must be carefully considered. First and foremost, all assets used in the scene must be both visually appealing and consistent with the overall aesthetic of the environment. Finding appropriate assets can be a time-consuming process, but it is essential to achieve a high level of realism in the projection.

Because the scenery is projected onto the walls of a large room, the quality and resolution of assets must be maintained at a high level. This presents a significant challenge, but it is essential to ensure that the projection remains visually impressive and immersive.

Anti-aliasing plays a critical role in creating a realistic environment. While Unity offers several anti-aliasing settings in the Project Settings, they come at a performance cost. A more effective solution is to rewrite the rendering shader with anti-aliasing functionality.

Lighting and shadows are also crucial considerations in achieving real-time, realistic rendering. To create scenes that look convincing under different lighting conditions and weather patterns, proper parameters must be experimentally determined and settled upon for each setting.

Particle effects, such as rain, clouds, smoke, and small, movable elements like stars and fireflies, are frequently used to enhance the immersive nature of the projection. However, the extensive use of particles can negatively impact runtime performance. As such, draw call

optimization should be performed to ensure that the particle effects are efficiently rendered without affecting performance.

Running the Python script via Unity

The Unity plugin for running Python code has a limitation when running an external Python script. While there are options to run hardcoded Python code within Unity, the more flexible approach is to simply call an external script, which can be reused easily, and allows for better readability. With Unity, to run an external script, the command below can be used.

```
using UnityEditor.Scripting.Python;

…

string scriptPath = Path.Combine(Application.dataPath,"my_script.py");
PythonRunner.RunFile(scriptPath);

…
```

The constraints to this approach however is that no additional arguments can be passed to the script. So, if the script requires additional arguments to be passed to it to work properly, it will fail. To overcome this constraint, the script was designed to search for an additional location for an argument, that is a file in the same directory with a specific name, i.e. "*scheme_option.txt*". The python script now has the following logic flow:

1. If an argument is passed to the script, take the passed value and prioritise it
2. If no argument was passed, search for a file called *scheme_option.txt*
3. If a file of the name is found, parse the contents and use the parsed data as the passed value
4. If neither is found, fail explicitly

Thus, if we wish to change the value that will be passed to the Helvar lights server, we simply need to change the contents of the *scheme_option.txt* file and call the script.