SitCoM - Situation Control Menu - User Manual

Niklas Suhre
December 20, 2023

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

List of Figures

List of Abbreviations

SitCoM Situation Control Menu

VR Virtual Reality

1 Workflow and Guidelines

Welcome to the Situation Control Menu (SitCoM)! The following pages will guide you through the usage of SitCoM in all its facets. Please read it thoroughly. If you find code written in *`Italic`*, please insert your own object names and values. The use of blanks is not recommended for names. Please pay attention to the upper and lower case at the beginning of placeholders. All placeholders that require the use of a name from before are color-coded. When the guidelines require you to insert some code "inside" e.g., a method or an `if`-condition, this means between the respective curly brackets (... { insert your code here } ...), as curly brackets mark the beginning and the ending of a code block.

1.1 Main Menu

Figure 1.1: Main Menu

This step is only necessary for the front-end-processes (s. 1.2 and 1.3) of adding scenarios and impacts, and the running of scenarios and impacts (s. 1.5). You do not need to start

SitCoM for the back-end-processes and the file exchange!

To start SitCoM just double-click the icon on the desktop or in the respective folder (s. 1.5). A new window opens, which shows the main menu of SitCoM (s. 1.1). Please note, the menus are only usable on a regular screen and not in Virtual Reality (VR). So, please perform all following actions before putting on a VR-headset (in case you are working on a VR-application).

1.2 Adding a new Scenario

Front End

Figure 1.2: Create Scenario

To add a scenario to the front end, go to the main menu (s. 1.1), and click the button that says “Create Scenario”. The respective menu (s. Fig. 1.2) will open. Now enter a distinctive and easy-to-remember name (“*Scenario Name in Front End*”) for the new scenario in the upper textbox. This name might include the position in the scene, as a new scenario is required for each and every new location. In the lower textbox you are able to enter a longer description for the scenario. This will be shown as a tooltip in the “Run Scenario”-Menu.

To finish the Front-End-Process click the “Create”-Button. The scenario will be written to a savefile, which can be found in the same folder as the application. This file is named

"scenarios.sitcom". It stores the progress regarding the (front end) creation of scenarios. These so-called "Savestates" can now be easily exchanged with other users or other workstations.

The "Cancel"-Button at the bottom of the menu brings you back to the main menu. Your input will not be saved!

Congratulations, you added a scenario to the front end!

Back End

The back end is the more complex part of adding a scenario. If SitCoM is running, please close the application (E.g., by pressing `Alt + F4`).

First, you need to download the 3D-models you want to use for your scene. There are many websites with free-to-use models, but of course you can also buy 3D-models. Then, open your project in Unity[1], and click on "Assets" in the menu bar. In the dropdown menu, click on "Import New Asset..." and select the files (resp. 3D-models) from the dialog menu.

Another option is to use the official Asset Store by Unity. There, select an asset, click "Add to My Assets" and then "Open in Unity". The Package Manager of Unity will open, there you can select your new asset and click on "Download". Then, you have to click on "Import" and on "Import" again in the new window. This will create a folder in your project, where your new asset is located. Please note, that you will have to use the same log-in for the Asset Store and Unity itself.

Please note, that the new models should not include their own camera view, as this will override the player's camera. Hence, the point of view will be altered by the object, which hinders a further use of the player.

Afterwards, you can move your new asset to a folder of your desire. This step, of course, is optional. A consistent folder-structure is set up in the *_SitCoM\Scenarios*-folder, where a new folder for every scenario can be created.

[1] Version: 2021.3.6f1

Next, create or open the scene you want the scenarios to take place in. Here, drag-and-drop your new asset to the scene. It will show up in the “Hierarchy” viewer (which is normally located on the left side of Unity). Of course, it would make sense, to already add it at its final position and in its final size - for the scene design and to get the coordinates, rotations, and scalings[2] - as this will be relevant in the following.

Normally, the added items should be imported as a so-called “Prefab”, which then can be accessed through the source code. In general, a “Prefab” is just a template for objects in the game that are added/instantiated during runtime. If the following steps do not work for you, please have a look into the chapter “Creating Prefabs” of the Unity Manual[3]. Delete the object from the scene after you are done, otherwise, it will always be visible - independently of the scenario it belongs to.

Now, still in Unity, go to the *_SitCoM\Scenarios*-folder[4]. Here, open the file called *SitCoM_ScenarioManager.cs* (s. Fig. 1.3). In a new window, Visual Studio will open the source code of the `SitCoM_ScenarioManager` class.

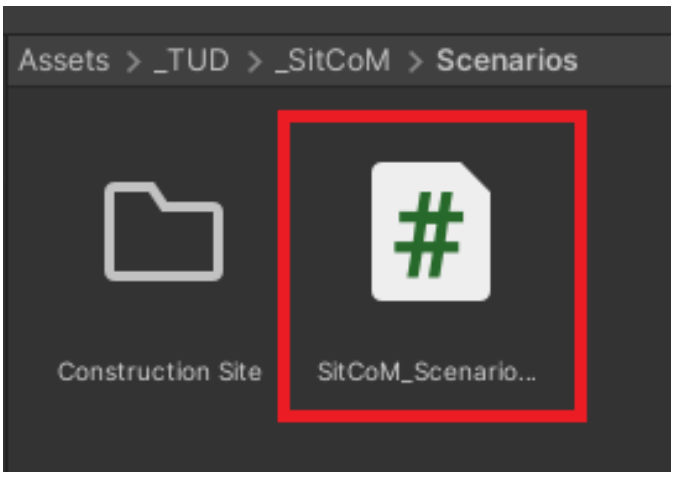

Figure 1.3: Folder of the SitCoM_ScenarioManager

[2] Shown in the Inspector Window
[3] `https://docs.unity3d.com/2021.3/Documentation/Manual/CreatingPrefabs.html` (visited on September 17, 2023)
[4] In the file manager at the bottom

Here, you have to do a few things. First, on top, but still inside the class (normally starting from line 6), you will have to declare the `GameObject`s you will use. This means, that you write the following line of code once for each and every 3D-model/item you want to use:

```
public GameObject distinctiveName;
```

Then, you will have to create a new method for spawning your scenario. Please use the following code snippet as a reference. In this code, you just include all `GameObject`s you have declared in the beginning and use the coordinates, rotation, and scaling you have found out before. This is done, by copying the five lines of code and changing the values of *itemName*, *distinctiveName*, and the numerical values. If you have many objects of the same kind, of course, you can include the code in a loop. As this is standard programming procedure, this is not covered here. You need the declaration of *distinctiveName*(s. above) only once per item type and not instance.

Please note, that Unity requires the coordinates, rotations, and scalings to be a so-called `float`, which is a special datatype for numbers. Therefore, just add the letter “f” at the end of every number. E.g., *503.7* becomes *503.7f*.

```
private void DistinctiveScenarioName()
{
   GameObject itemName= Instantiate(distinctiveName);
   itemName.transform.position = new Vector3(x-coordinate,y,z);
   itemName.transform.Rotate(x-rotation,y,z,Space.Self);
   itemName.transform.localScale = new Vector3(x-scale,y,z);
   spawnedObjects.Add(itemName);
}
```

Still in the `SitCoM_ScenarioManager` class, find the `Update()`-method in the code. There, you will see a few `if`-conditions. Find the one, that says:

```
if(AreImpactsCurrentlyHappening())
{
   ...
}
```

Inside this `if`-condition add the following code snippet according to your needs. There may be already other `if`-conditions you can just copy and edit accordingly.

```
if(IsScenarioIncluded("Scenario name in Front End"))
{
    DistinctiveScenarioName();
}
```

Last but not least, save the class-file and go back to the main window of Unity. On the left-hand side, in the hierarchy viewer, select the “SitCoM_ScenarioManager” (indicated by a box icon). If you do not see it, you need to assign the script to an object in the scene, which is existent over the entire (!) runtime of the scene (just by dragging-and-dropping the script onto the object; e.g., just place a 3D-model very far away from your regular scene and assign the script to this). Normally on the right-hand side, you will find the “Inspector”. On top of the Inspector you will find all the (as `public`) declared objects from the source code (s. Fig. 1.4). Here, for all of your objects click the circle on the right side of each line and select the respective asset from the dialog (Tab: “Assets”). If the asset does not appear here, it is not declared as a “Prefab” (s. 1.2).

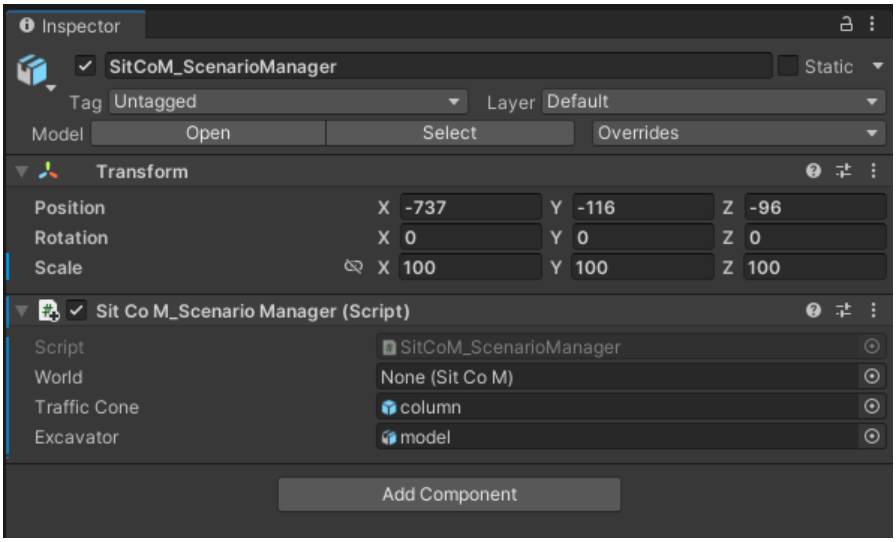

Figure 1.4: SitCoM_ScenarioManager in Unity Inspector

Congratulations, you added a scenario to the back end!

1.3 Adding a new Impact

Front End

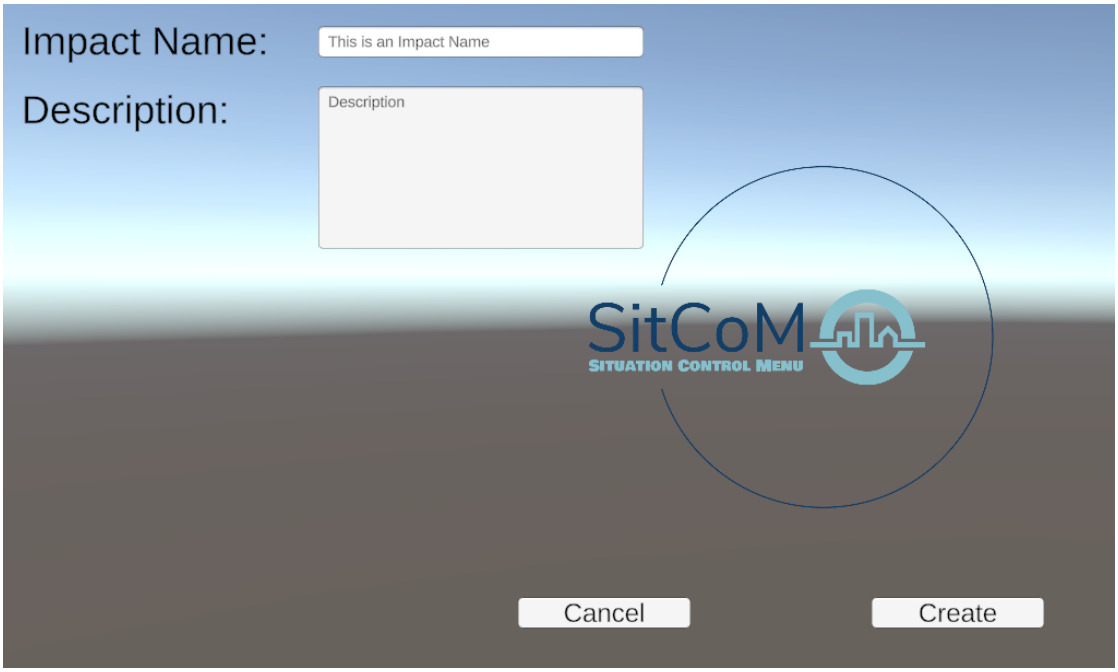

Figure 1.5: Create Impact

To add an impact to the front end, go to the main menu (s. 1.1), and click the button that says “Create Impact”. The respective menu (s. Fig. 1.5) will open. Now enter a distinctive and easy-to-remember name (“*Impact Name in Front End*”) for the new impact in the upper textbox. In the lower textbox you are able to enter a longer description for the impact. This will be shown as a tooltip in the “Run Scenario”-Menu.

To finish the Front-End-Process click the “Create”-Button. The impact will be written to a savefile, which can be found in the same folder as the application. The file is named *“impacts.sitcom”*. It stores the progress regarding the (front end) creation of impacts. This enables you, to exchange these so-called “Savestates” with other users or other workstations.

The “Cancel”-Button at the bottom of the menu brings you back to the main menu. The input will not be saved!

Congratulations, you added an impact to the front end!

Back End

The back end is the more complex part of adding an impact. If SitCoM is running, please close the application (E.g., by pressing `Alt + F4`).

The prerequisite for the following steps is that you already have an object (E.g., a 3D-model) in the scene, that will be affected. If this is not the case, please refer to chapter 1.2 on how to integrate new objects to the scene. In the following, the line numbers are referring to an empty class. This means, they are not applicable for the changing of existing classes.

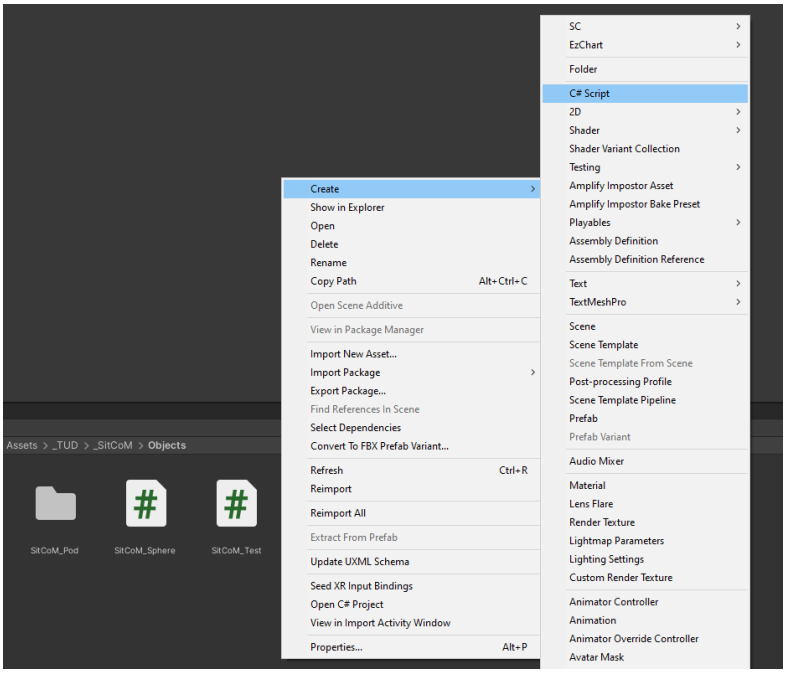

Figure 1.6: Adding a new Script

Open the project in Unity[5], and navigate to the _*SitCoM**Objects*\-folder. Right-click an empty space in the file manager and go to "Create" → "C# Script" (s. Fig. 1.6). Enter a distinctive and easy-to-remember name for your new class ("*ClassName*"). Double-click the new script - Visual Studio will open.

First, change the following line of code (normally line 5):

```
public class ClassName: MonoBehaviour
```

to:

```
public class ClassName: SitCoM_Object
```

Find the `Start()`-method and add the following line of code inside the method (normally line 10):

```
this.StartUp();
```

Inside the `Update()`-method add the following code (normally line 16):

```
if(AreImpactsCurrentlyHappening())
{
    if(IsImpactIncluded("Impact name in Front End"))
    {
        ...
    }
}
```

Inside the inner `if`-condition you can now code the behavior of the object during an impact. If you have multiple impacts, that effect the object, just copy the inner `if`-condition and change the *Impact name in Front End*. As these behaviors differ greatly between different objects, I will not cover how to implement all kinds of different impacts here. Please refer to standard programming and Unity manuals for further information and troubleshooting.

The final step is, that you need to save and add your script to the respective representation in the virtual city. Just drag-and-drop the script onto the object or add the script as a component of the object via the Unity Inspector.

Congratulations, you added an impact to the back end!

[5]Version: 2021.3.6f1

1.4 Creating an Executable Version of SitCoM

In Unity's menu bar, click "File" → "Build Settings". In the new window (s. Fig. 1.7), click "Build" which is located at the bottom. Select a folder from the dialog, where you want SitCoM to be saved, and confirm. The build will take a few seconds, while the executable version of SitCoM is being generated. The content of the new folder can be seen in Fig. 1.8.

Please note, that the scenario and impact files are not automatically generated through the build but at runtime of SitCoM. You can also paste scenario and impact files before running SitCoM (s. 1.6).

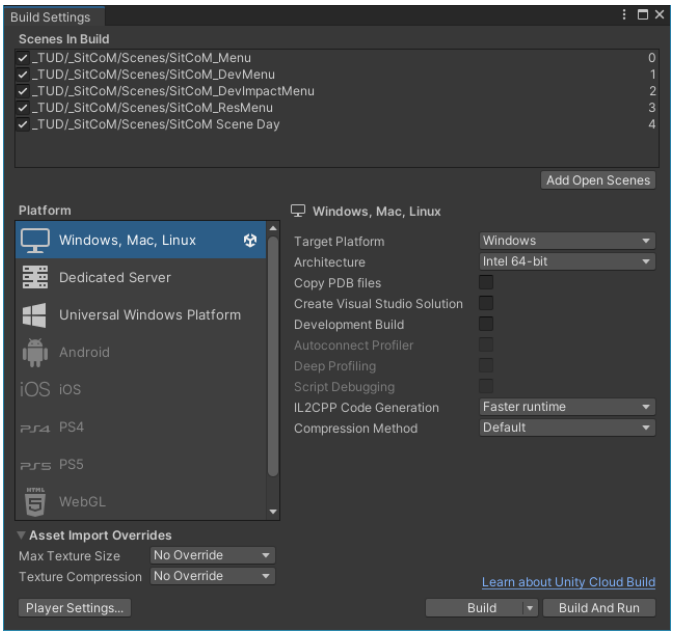

Figure 1.7: Build Settings

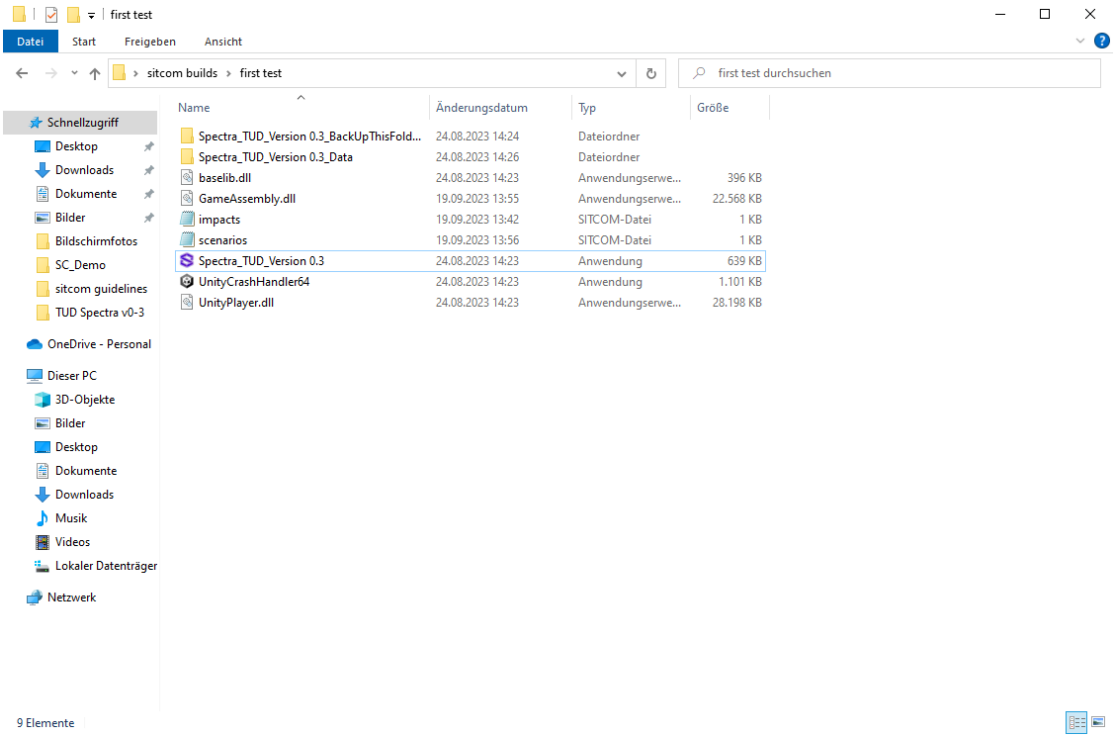

Figure 1.8: Build Folder

1.5 Running Scenarios and Impacts

Go to your build folder (s. 1.4) and start the application. The main menu of SitCoM will open (this does only work on a regular display and not in VR).

In the main menu (s. Fig. 1.1) click the button that says “Run Scenario”. The menu for the research stage will open. See Fig. 1.9 for an overview.

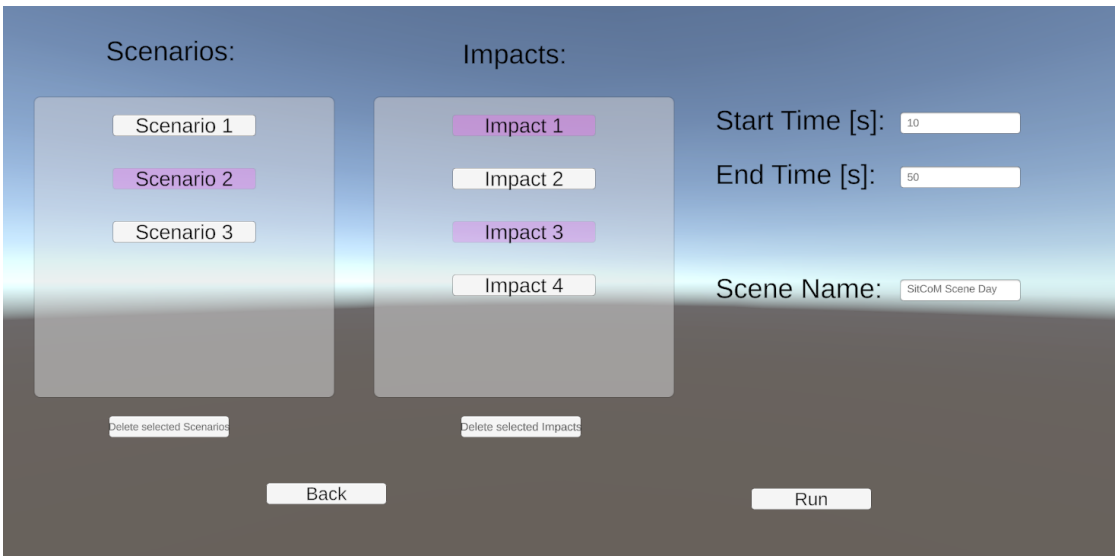

Figure 1.9: Run Scenarios and Impacts, the name of the application can vary

On the left-hand side you will find the scenarios you have added to the front end (s. 1.2). In the middle column there are the impacts you have added to the front end as well (s. 1.3). Both lists are sorted alphabetically.

The selection of scenarios and impacts you want to add to your scene is very easy. Just click on the desired scenario/impact, it will change the background to a pinkish color. This color change shows, that the selection has been successfully made. Clicking the button again, will deselect the scenario/impact. This is also indicated by a color change back to white.

Below the two “selection panels” you will find buttons for deleting scenarios and impacts. If you click them, the scenarios or impacts you have selected will be deleted permanently. Be careful with these actions, as this cannot be undone. If you delete a scenario or an impact by accident, you will have to add it to the front end again.

On the right-hand side, you can enter a start time and an end time for the scenarios and impacts. Please note, that the starting and ending of all scenarios and impacts are carried out simultaneously. So, it is not possible to select different starting and end times for single scenarios or impacts. If you do not enter a start time, the scenarios and impacts will start immediately after the scene has been loaded. If you do not enter an end time, the

end time will be set to the maximum value of an integer in C#, which is $2,147,483,647$. This is also the maximum you can enter for both times.

On this side, you have to enter the name of the scene you want to run as well. This scene has to be added to the build. You achieve this by opening your scene, going to "File" $\rightarrow$ "Build Settings" $\rightarrow$ "Add Open Scenes", and ticking the new scene (if not automically done). Close the "Build Settings". Now, if you enter the scene name during runtime, the scene should open, if not, check for spelling mistakes, or if you have added the scene correctly to the build.

If you are happy with the selections and inputs you have made, click the "Run" button. Your scene will open and the scenarios and impacts should start. You can also put on the VR-headset, if you are using one.

Congratulations, you are now running scenarios and impacts!

1.6 File and Version Exchange

For the file exchange go to your build folder (s. 1.4). SitCoM creates two different savefiles, one for scenarios (*"scenarios.sitcom"*), and one for impacts (*"impacts.sitcom"*).

Just copy or paste your savefiles into the first level of your build folder (s. Fig. 1.8) and run the application. The scenarios and impacts should now be visible in the "Run"-menu (s. 1.5).

Please note, that the file name should not be changed, as SitCoM is currently not able to handle different file names. Also, be careful with overwriting over savefiles, as this data is then lost.

To exchange the SitCoM version between different workstations, there are mainly two approaches. One is by using Github and building the project on the other workstation again. The other and easier way is by just copying the build folder onto the second workstation, as the new executable version is already included in this folder. I recommend doing both, to always have the code up-to-date, as this is relevant for the back end processes.

1.7 Troubleshooting

In the following, I will describe the main problems I had during the implementation of SitCoM and how I fixed them. This should help other programmers and users when working with Unity in general and on SitCoM in particular.

Buttons in Menu not working

Problem: The buttons in the menu are not reacting to the user's input. Neither do they change color, when the mouse is hovering over them, nor are they clickable.

Solution: Make sure, the "Interactable" property of the button is turned on. Check, if an "EventSystem" is part of the scene as well, as this is required for event handling.

Writing and Loading

Problem: The scenarios and impacts have to be written and loaded multiple times in multiple menus. It is inefficient, to implement this multiple times.

Solution: The writing and loading process is implemented as part of the `SitCoM` class, as this is available in all menus (and their respective scenes). This is due to the implementation of the "shifting" of `SitCoM` between the scenes. The object, and therefore its connected "information" stay the same and are not instantiated in every scene again but only in the main menu.

User Pop-Up Dialogs

Problem: The dialogs provided by the `UnityEditor` (s. Fig. 1.10 for an exemplary user dialog) are not available outside the editor, hence, are not suitable for use once the program is released.

Solution: By the use of C# directives (E.g., `#if [...] #endif`) the use of dialogs is constrained outside the editor. Hence, the dialogs will not show up. This results in a decreased user experience, as the users now have to be aware of their actions, before pressing certain buttons. A different solution, with user dialogs, should be intended for future versions.

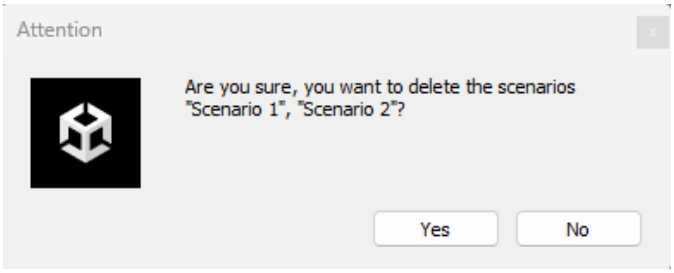

Figure 1.10: User Dialog for Deleting a Scenario

Implementation of Impacts

Problem: When the consequences of an impact are implemented in an own method, which is controlled by the timer of `SitCoM_Object`, Unity is blocking all changes. This is due to the attempt of a cross-thread action, which Unity does not support.

Solution: The implementation of an impact has to be carried either in the `Update()` method or an external function that is called from the inside of `Update()`. Then, everything is executed in one thread only, which is no problem for Unity. One disadvantage is, that the “activation" of an impact is still controlled by the timer implemented in `SitCoM_Object`. But, the delay should not be noticeable to the user, as the temporal resolution is set to $\frac{1}{100}$ of a second.

Acknowledgements

This thesis has been made possible through the support of NUMENA and Spectra Cities.

Funded by the European Union (ERC, scAInce, 101087218). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.