D8.9: Training Material

Dissemination level:	Public
Document type:	Report
Version:	1.0.0
Date:	August 24th , 2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement #769553. This result only reflects the author's view and the EU is not responsible for any use that may be made of the information it contains.

Document Details

Project Number	769553
Project title	Council of Coaches
Title of deliverable	Training Material
Due date of deliverable	August 31st, 2020
Work package	WP8
Author(s)	Harm op den Akker (RRD), Daniel Davison (CMC), Fajrian Yunus (SU), Tessa Beinema (RRD), Stephanie Kosterink (RRD), Marian Hurmuz (RRD), Randy Klaassen (CMC), Gerwin Huizing (CMC), Mark Snaith (UDun)
Reviewer(s)	Jorien van Loon (CMC)
Approved by	Coordinator
Dissemination level	PU: Public
Document type	Report
Total number of pages	86

Partners

- University of Twente – Centre for Monitoring and Coaching (CMC)
- Roessingh Research and Development (RRD)
- Danish Board of Technology Foundation (DBT)
- Sorbonne University (SU)
- University of Dundee (UDun)
- Universitat Politècnica de València, Grupa SABIEN (UPV)
- Innovation Sprint (iSPRINT)

Abstract

This deliverable contains an overview of all the "training activities" performed within the context of the Council of Coaches project – those activities that are meant to teach the reader/listener how to apply the knowledge and tools from the Council of Coaches project in their own work. This includes links to published work (e.g. webinars, workshop) as well as software documentation and tutorials.

Table of Contents

List of figures

List of tables

Symbols, abbreviations and acronyms

ASAP	Articulated Social Agents Platform
BML	Behaviour Mark-up Language
CMC	Centre for Monitoring and Coaching
COUCH	Council of Coaches
D	Deliverable
DBT	Danish Board of Technology Foundation
EC	European Commission
ISPRINT	Innovation Sprint
M	Month
MS	Milestone
RRD	Roessingh Research and Development
RRI	Responsible Research and Innovation
SU	Sorbonne University
UDun	University of Dundee
UPV	Universitat Politècnica de València
UT	University of Twente
WoZ	Wizard of Oz
WP	Work Package

1 Introduction

Council of Coaches is a project that is funded through the European Commission, and thus indirectly by the European tax payers. As such, as a consortium, we find it very important to (1) generate as much as possible openly available project outcomes, and (2) take appropriate actions to make sure these results reach their target audience and are understood well enough to be used.

This public document serves as a guide that covers all the activities performed and documentation generated within the Council of Coaches project that helps to "train the public" in applying the results and outcomes of the Council of Coaches project. One of the major outcomes of the Council of Coaches project is its open source embodied conversational agents platform – Agents United[1]. This platform is a highly innovative technical software platform that allows 3rd party application developers to use the combined power of the embodied conversational agent platforms GRETA (Sorbonne University) and ASAP (University of Twente) in combination with the Dialogue and Argumentation Framework (University of Dundee) and the WOOL Dialogue Platform (Roessingh Research and Development). The main focus of the training material developed within the project is on this software platform.

Section 3 lists the software documentation generated to train new users in setting up and using the Agents United platform.

Although the Agents United platform as a whole is the major technical innovation from the Council of Coaches project – its major core platform components (WOOL, ASAP, GRETA, and DAF) can also be used individually. Whereas ASAP and GRETA have a long history of development, pre-dating the Council of Coaches project, the WOOL Dialogue Platform was specifically developed for Council of Coaches, and the Dialogue and Argumentation Framework has seen a large boost in maturity.

Section 4 provides documentation (or links to online documentation) for the four major individual platform components of the Agents United platform.

Besides software documentation, the project has organized several training activities (outside those listed in other deliverables) in the form of webinars and workshops.

Section 5 lists the webinars and workshops organized by the Council of Coaches project not yet described in other deliverables.

Finally, Council of Coaches is a highly innovative concept and as such provides a good context for educational activities like courses or individual student assignment – in order to train the next generation of innovators.

Section 6 provides an overview of how Council of Coaches was used in various educational activities.

[1] See http://www.agents-united.org/

2 Objectives

This document provides an overview of the major training materials produced in the context of the Council of Coaches project.

3 Agents United Platform Documentation

The release of the open source Agents United Platform is a major outcome of the Council of Coaches project. In order to ensure a proper uptake of the technology released as part of this platform, it is essential that proper documentation and training material is provided. The target population for this documentation are highly skilled software developers, that can use (parts of) the platform to create their own multi-party embodied conversational agent tools and services.

The software of the Agents United Platform is hosted on GitHub, a platform that hosts most of the world's open source software solutions. The GitHub platform is used to host the actual code, but provides many additional tools and services around the hosted software, including issue tracking, documentation and release management.

Documentation should be closely packaged with the software itself, and therefore most of the software documentation can be found on GitHub as part of the set of software repositories released under the Agents United Platform GitHub Page: https://github.com/AgentsUnited

The different repositories are listed in Table 1 below.

Table 1: List of repositories under the Agents United GitHub page.

Repository	Description	Link
architecture	Models and views defining the architecture of Agents United	https://github.com/AgentsUnited/-architecture
topic-selection-engine		https://github.com/AgentsUnited/topic-selection-engine
daf	The Dialogue and Argumentation Framework module	https://github.com/AgentsUnited/-daf
demonstrator	Start here. This repository centralizes the other modules of Agents United into a runnable demonstrator.	https://github.com/AgentsUnited/-demonstrator
dialogue-gui		https://github.com/AgentsUnited/dialogue-gui
documentation	A collection of additional documentation and images / screenshots / videos related to the agents-united platform.	https://github.com/AgentsUnited/documentation
intent-planner	The Conversational Intent Planner module	https://github.com/AgentsUnited/intent-planner
unityproject	Unity3D scenes for the agents	https://github.com/AgentsUnited/-unityproject
universaal	Modules for connecting to the universAAL IoT platform	https://github.com/AgentsUnited/-universaal

In the following subsections, we include the documentation on how the modules represented in the repositories listed in Table 1 can be used. This material is a snapshot of the work at the moment of the

writing of this deliverable. Documentation and the generation of additional support material for the Agents United platform will continue after the project's end.

Consult the Agents United GitHub pages at www.github.com/agentsunited for the latest version of the following documentation.

3.1 Architecture

The **architecture** repository does not contain any software that is part of the Agents United platform, but rather contains the "official" formal representation of the software architecture that was used in the design of the platform and is documented elsewhere in e.g. D7.1.

Some notes from the repository documentation:

Architecture project and models created with Enterprise Architect.

- The file councilofcoaches.eap contains the entire Enterprise Architect project.
- The Web folder contains a browsable HTML export of the project. It appears the scripts are no longer supported by Chrome nor Firefox, but Microsoft Edge still loads them.
- The Diagram folder contains image exports of all the diagrams in the project.

All original content licensed under the Apache Software License 2.0.

Below in Figure 1 an example is given of the diagrams contained in the **architecture** repository.

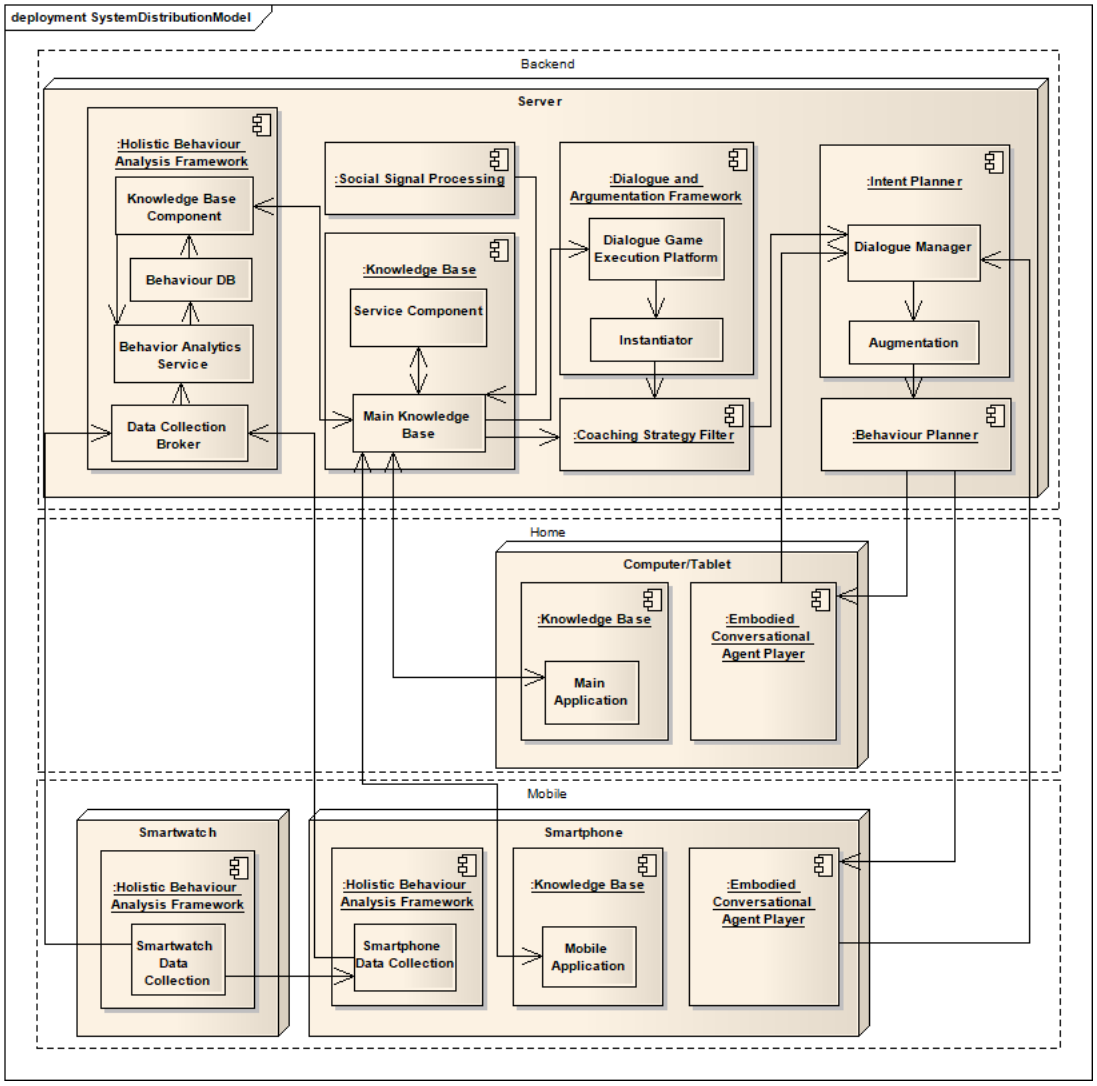


Figure 1: Example of the architecture diagrams from the Agents United architecture repository: the System Distribution Model.

3.2 Topic Selection Engine

The Topic Selection Engine is a module for the Agents United platform. The module allows to automatically select a conversational topic that is most relevant to be discussed given the available information. The topic selection algorithm will make this choice based on a topic structure - a tree of topic nodes - and selection parameters that can be added to the topic nodes in that structure.

N.B. WOOL Web Service needed.

The Topic Selection Engine needs access to e.g. an instance of the WOOL Web Service (see www.woolplatform.eu) since it retrieves the values for the selection parameters from the WOOL Web Service.

3.2.1 Install and run the Topic Selection Engine

Requirements:

JAVA

The web service was tested with OpenJDK 11.

https://jdk.java.net/archive/

Download the ZIP file and extract it to a location on your hard disk. For example: C:\apps\jdk-11 Then set environment variable JAVA_HOME. For example: C:\apps\jdk-11 And add the bin directory to your path. For example: C:\apps\jdk-11\bin

You should now be able to run Java from the command prompt. For example:

- *java -version*

TOMCAT:

The web service was tested with Tomcat 8.5.

http://tomcat.apache.org/download-80.cgi

Download the 32-bit/64-bit Windows Service Installer of Tomcat 8.5. Run the installer and check the following pages: Choose Components:

- You may want to add Tomcat > Service Startup so Tomcat is started automatically when the computer is started.
- Add "Host Manager". This is needed to deploy web applications from the Gradle build file. Configuration:
- At Tomcat Administrator Login fill in a user name (for example "admin") and password.
- Set "Roles" to: admin-gui,manager-gui,admin-script,manager-script The script roles are also needed for deployment from the build file. Java Virtual Machine:
- Select the directory to the JDK. For example: C:\apps\jdk-11

Configuration of the JDK location, Java options and memory size:

- From the Windows start menu, open Monitor Tomcat. If you get a permission error, you may need to open the Tomcat folder in Windows Explorer first: C:\Program Files\Apache Software Foundation\Tomcat 8.5 The Tomcat monitor opens in the notification area of the task bar.
- Open the Tomcat monitor from the task bar.
- Open the tab Java.

You should now be able to open the Tomcat manager at: http://localhost:8080/

CONFIGURATION:

The web service is configured with this file that needs to be created: \CoachingEngine\gradle.properties

You can make a copy of this file and change it: \CoachingEngine\gradle.sample.properties

DEPLOY:

Make sure that Tomcat is running.

The web service is deployed with Gradle task cargoRedeployRemote. It requires the following properties that you need to fill in gradle.properties:

- tomcatDeployPath
- remoteTomcat*

Open a command prompt in: \CoachingEngine Enter this command:

.\gradlew build cargoRedeployRemote

If you want to make a clean build and deploy, then enter:

.\gradlew clean build cargoRedeployRemote

After deploying you can access the Swagger interface at: http://localhost:8080/servlets/coaching-engine

USAGE:

Also see the Swagger interface that should be available (after deployment) at: http://localhost:8080/servlets/coaching-engine

First, retrieve an auth token from the WOOL Web Service with POST /auth/login. The request body should be a JSON object like this:

{ "user": "user@example.com", "password": "p4ssw0rd", "tokenExpiration": 1440 }

The "user" is case-insensitive. The "tokenExpiration" is optional. It can be a value in minutes or "never". The default is 1440 minutes (24 hours).

The response is a JSON object like this:

{ "user": "user@example.com", "token": "eyJhbGciOiJI..." }

The "user" in the response may have different case than the case-insensitive "user" in the request.

Include this "user" and "token" in your calls to the Coaching Engine endpoints.

3.2.2 Defining a new topic structure and selection parameters

-- To be added.--

3.2.3 License

The Topic Selection Engine (TSE) is licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0).

3.3 Dialogue and Argumentation Framework (DAF)

To demonstrate the Dialogue and Argumentation Framework (DAF), it has been bundled with a web-based interface that allows a user to control the selection of dialogue moves provided by the framework. The demonstrator also allows for the editing and creation of dialogue protocols, creation and editing of content for use in dialogues, and the creation and editing of mock coaching variables for testing purposes.

3.3.1 Build

1. The DAF uses Docker, including docker-compose.
2. Download or clone this repository.
3. Open a command line terminal and go to the root folder of the repository
4. Run the command docker-compose up. The containers will start to build. This process may take several minutes. During this time, you may see output in the terminal relating to a ConnectionRefusedError; this is normal.
5. When the DAF is ready, the following line will be printed in the terminal daf_controller | Dialogue and Argumentation Framework ready.

3.3.2 Run

1. The docker-compose up command also starts the DAF. After the first time installation it should be faster.
2. To test the DAF demonstrator, open a web browser and enter the URL [http://localhost:8080]
3. When the page has loaded, select one of the four options: *Test a protocol*, *Edit protocols*, *Edit content*, or *Edit coaching variables*.

3.3.3 Test a protocol

1. Select a protocol from the drop-down menu and follow the instructions.
2. After entering the name of the user, you will be presented with a simulation of the dialogue. You can select both the agent and the user moves and examine how the dialogue unfolds and progresses.

3.3.4 Edit protocols

1. Select a protocol from the drop-down menu, or choose New protocol.
2. Edit or create the protocol and click Save; if you are creating a new protocol, be sure to enter a name in the box.

3.3.5 Edit content

1. Choose either Edit argument models or Edit dictionary.
2. Create or edit the content when it is displayed in the box.

3.3.6 Edit coaching variables

The interface to edit coaching variables is provided to allow dialogues to be tested independent of the Knowledge Base. For this demonstrator, coaching variables are expressed in a JSON object as key-value pairs.

3.3.7 Stopping

Use ctrl+c to stop the DAF terminal and then the command docker-compose down to stop DAF from Docker.

3.3.8 License

Dialogue and Argumentation Framework (DAF) is licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0).

3.4 Demonstrator

The Agents United Open Platform allows you to build a system of multiple virtual embodied conversational agents. This repository contains a demonstrator af several coaching agents for different health domains. The Agents United platform and this demonstrator are the outcome of the Council of Coaches European research project.

3.4.1 Repositories

Each of the different modules that compose the Agents United platform has its own repository. Some modules are hosted and maintained here in Agents United while others are external, developed and hosted in 3rd party repositories.

Modules hosted in Agents United:

- DAF: The Dialogue and Argumentation Framework modules
- Intent-Planner: The Conversational Intent Planner modules
- UnityProject: Unity3D scenes for the agent's user interface
- Demonstrator: This repository, which also contains a collection of executable scripts
- Topic-Selection-Engine: The Topic Selection Engine module
- universAAL: Modules for connecting to the universAAL IoT platform

Other modules hosted elsewhere:

- Greta: Socio-emotional virtual characters for agents by ISIR - University of Sorbonne (see: https://github.com/isir/greta);
- HMI Build: Multi-platform build system by HMI - University of Twente (see: https://github.com/ArticulatedSocialAgentsPlatform/hmibuild);
- WOOL Web Service: Knowledge base and dialogue management web service of the WOOL Platform (see: https://github.com/woolplatform/wool/tree/master/java/WoolWebService).

3.4.2 Architecture

If you are interested in figuring out how the entire Agents United platform works you can check out the architecture documentation and diagrams in the Architecture repository (see Section 3.1).

3.4.3 License

The modules hosted in Agents United are licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0) except when otherwise is stated. External modules linked here have their own licenses.

3.4.4 Instructions

The following instructions will guide you through setting up and running the demonstrator. You can find a list of the latest hotfixes and solutions for possible problems you may come across at the end of this Readme.

3.4.4.1 Pre-requisites

- **Windows 10 TTS Voices** (Instructions here are only for Windows 10 Pro)
  - You will need the default US voices for Text-To-Speech: Mark, David, and Zira. These can be installed from Settings, Text-to-Speech. Check this wiki for further instructions and troubleshooting: https://github.com/hmi-utwente/HmiASAPWiki/wiki/MS-API-Voices
- **Unity3D** (version 2017.4.24f1. Other 2017.x versions might work, versions 2018.x or later do NOT).
  - In the installer, include the Android, iOS, tvOS, macOS build supports
- **Docker** (Should work with any current version of Docker, including Community for Windows)
  - Select Linux containers during installation
- **Java 8 JDK** (Update 211)
  - Make sure you add the Java bin folder to your PATH environment variable.
  - Make sure you set the JAVA_HOME environment variable.
- **Ant build system** (Latest version)
  - Make sure you add the Ant bin folder to your PATH environment variable.
  - Make sure you set the ANT_HOME environment variable.
- **Visual C++ Redistributable for Visual Studio** (versions 2013 and/or 2015)
  - This is recommended for Unity, although your Windows installation may already have this
  - Approximately 10GB of disk space and at least 8GB RAM (recommended)

3.4.4.2 Installation

The following instructions should be followed in order to install the Council of Coaches technical demonstrator.

Clone the repository

- Note that on Windows the longpaths parameter needs to be set, using `git config --system core.longpaths true`
- Clone the repository: `git clone https://github.com/AgentsUnited/demonstrator.git` (we will refer to its folder as `{demonstrator}`)

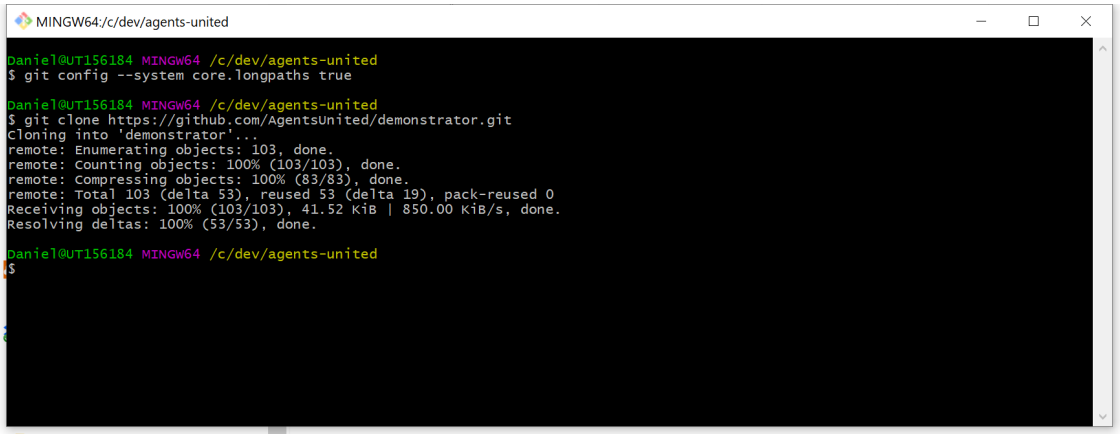


Figure 2: Cloning the Demonstrator repository.

- Init and get the linked submodules with git submodule update --init –recursive

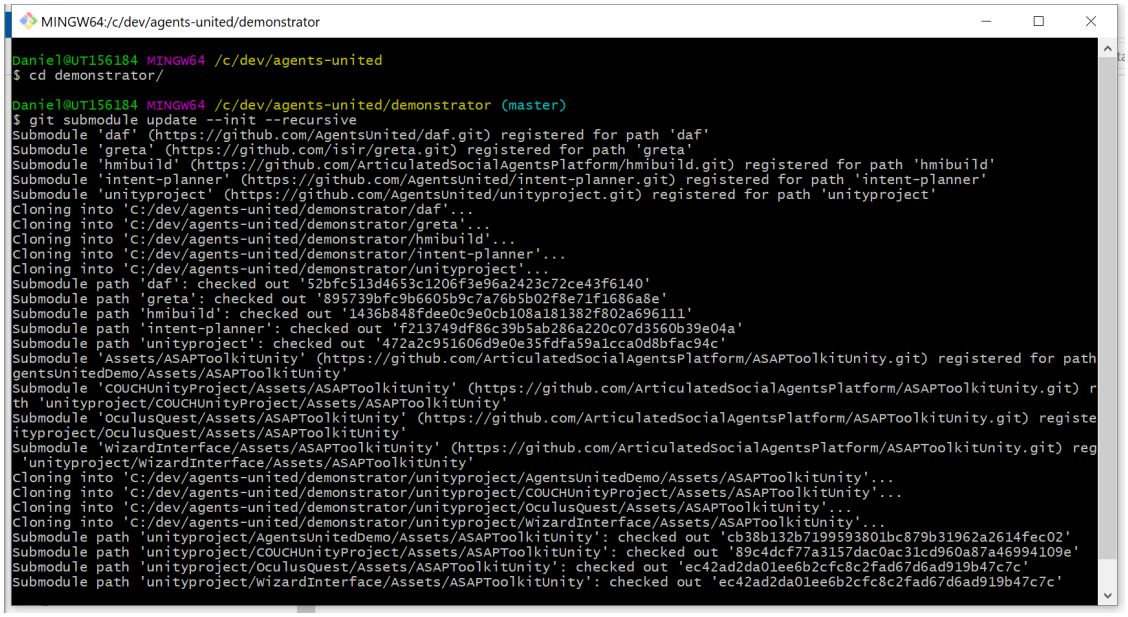


Figure 3: Retrieving the Demonstrator submodules.

- Instead of using Git, you can manually download the code. If you do so, download and place the submodules as well.

Setup the Dialogue and Argumentation Framework (DAF)

1. Start Docker. Right click the tray icon and go to Settings. Go to Shared Drives and share the main drive. Go to Advanced and set Memory to 4GB (Recommended).
2. Open a command line shell, go to `{demonstrator}\daf` and type the command `docker-compose pull`
3. Then type the command `docker-compose build` to build the various containers. When finished it should look similar to this:

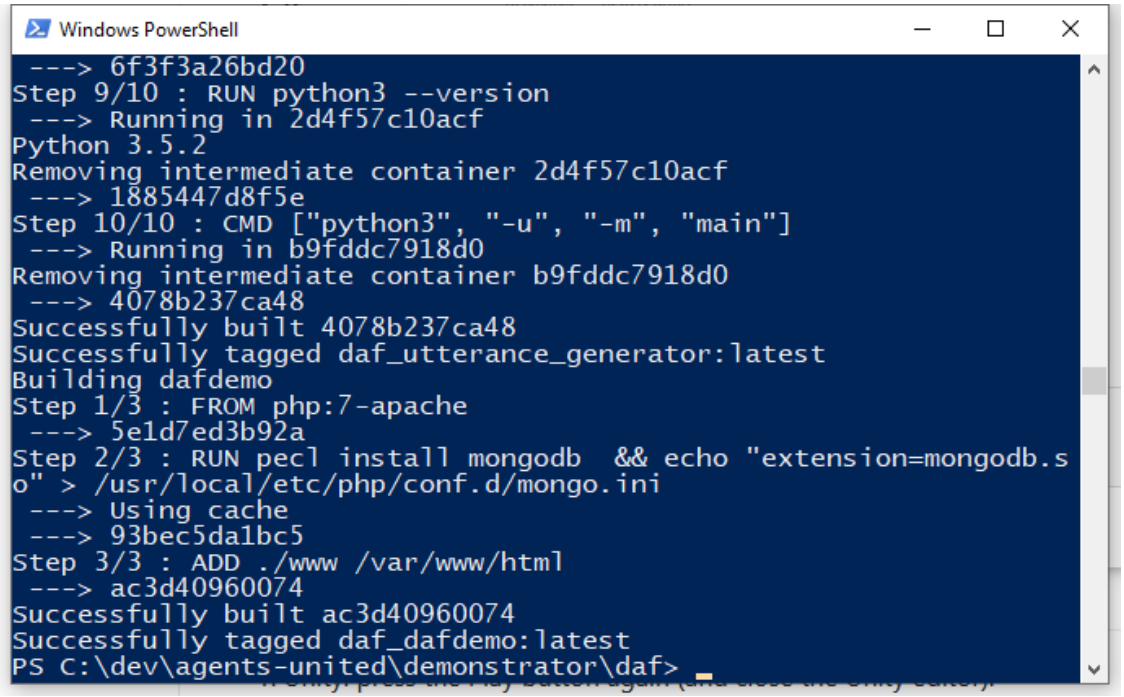


Figure 4: Setting up the Dialogue and Argumentation Framework.

Setup ASAP and the Intent Planner

1. Open a command line shell, go to `{demonstrator}\intent-planner` and execute the following commands:
2. `ant clean`
3. `ant resolve` - this downloads all dependencies for this project (including libs native to your operating system, 32/64 bit). This process may take a while. When finished it should state `BUILD SUCCESSFUL` without any errors.

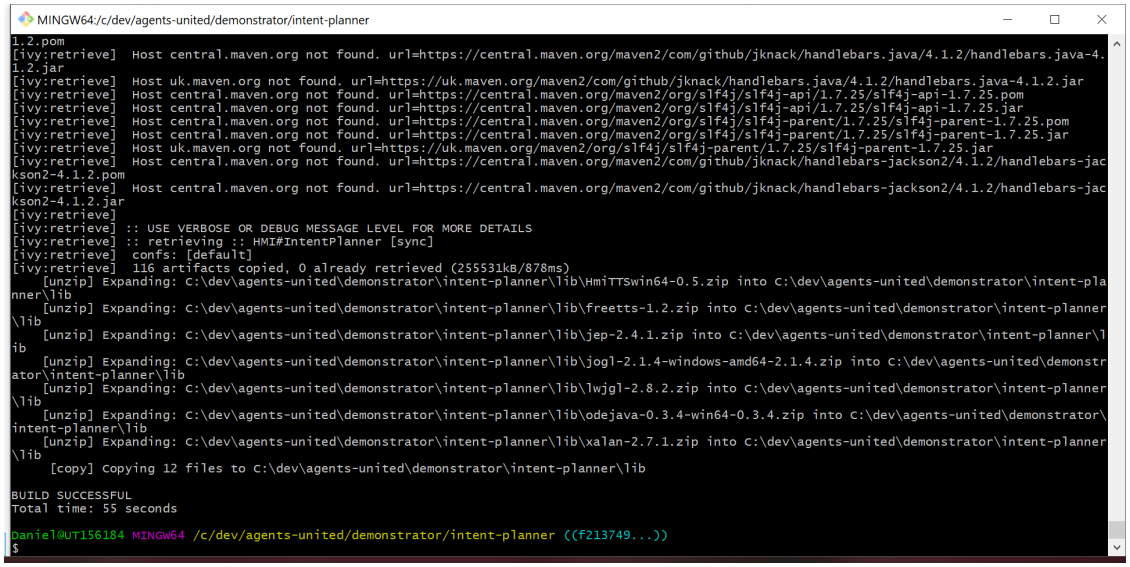


Figure 5: Setting up ASAP and the Intent Planner.

4. `ant compile` - this compiles all java source files.

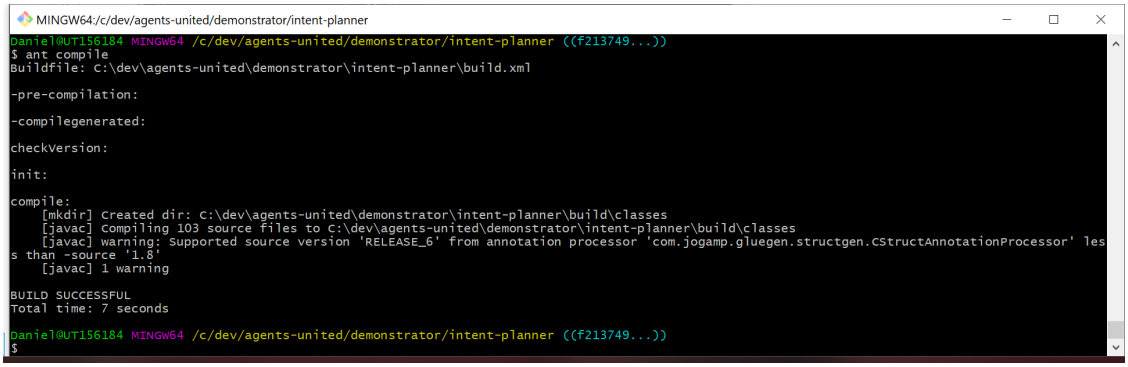


Figure 6: Compiling all code for ASAP and the Intent Planner.

Install Mary TTS

1. Download *Mary TTS* from http://mary.dfki.de/download/index.html, Runtime Package, and unpack the contents in any folder you want. We will refer to this folder as `{marytts}` from now on.
2. Go to `{marytts}/bin` and run `marytts-component-installer.bat`. From that tool, install the following languages:
    a. `enUS/cmu-slt`
    b. `en-US/cmu-bdl`
    c. `fr/enst-camille`
    d. `fr/enst-camille-hsmm`

Setup Greta agents

1. Open a command line shell, go to `{demonstrator}\greta` and execute the command `ant build`
2. Go to `{demonstrator}\greta\bin` and edit the files `vib.ini` and `Modular.xml` to replace `./Environments/Empty.xml` with `./Environments/Projects/Council of Coaches/TechnicalDemonstrator.xml`.
3. Also in `vib.ini`, replace `<MARY_SERVER_DIRECTORY>` with `{marytts}\bin`.

Setup the Unity scene

1. Start Unity. Select Open project, and then select the folder `{demonstrator}\unityprojet\AgentsUnitedDemo`. (You may get a warning dialog depending on your exact version of Unity. Ignore it and Continue).
2. In the Project assets panel (usually bottom-left), navigate to `\Assets\AgentsUnited\Scenes` and double-click the scene `MainScene.unity`. Unity will now import and set up all assets for your system (this may take a while). When finished your scene in the editor should look similar to this:

Figure 7: The Unity scene after successful installation.

3.4.4.3 Running

The following instructions should be followed for running the Council of Coaches technical demonstrator.

Run OpenMary TTS

Open a command line shell, go to `{marytts}\bin` and execute `marytts-server.bat`. Wait until it is up and running on port 59125

Run the Dialogue and Argumentation Framework (DAF)

1. Start Docker
2. Open a command line shell, go to `{demonstrator}\daf` and type the command `docker-compose up`. Wait until it is up and running. The console should print `Dialogue and Argumentation Framework ready` and look similar to this:

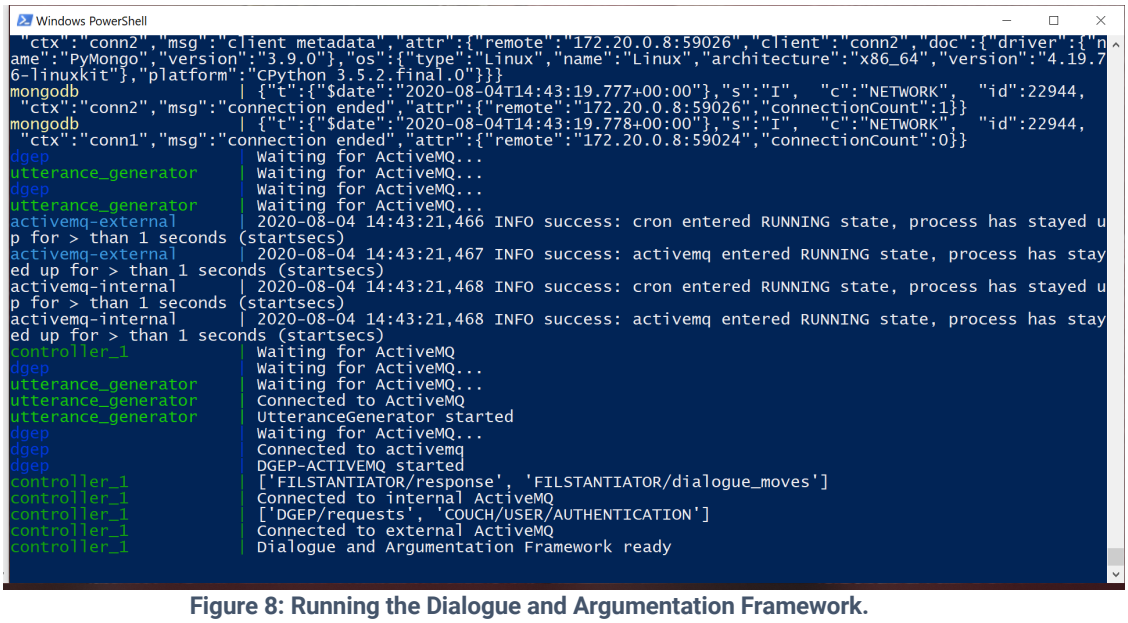


Figure 8: Running the Dialogue and Argumentation Framework.

If you prefer to use the Docker dashboard, it should look similar to this:

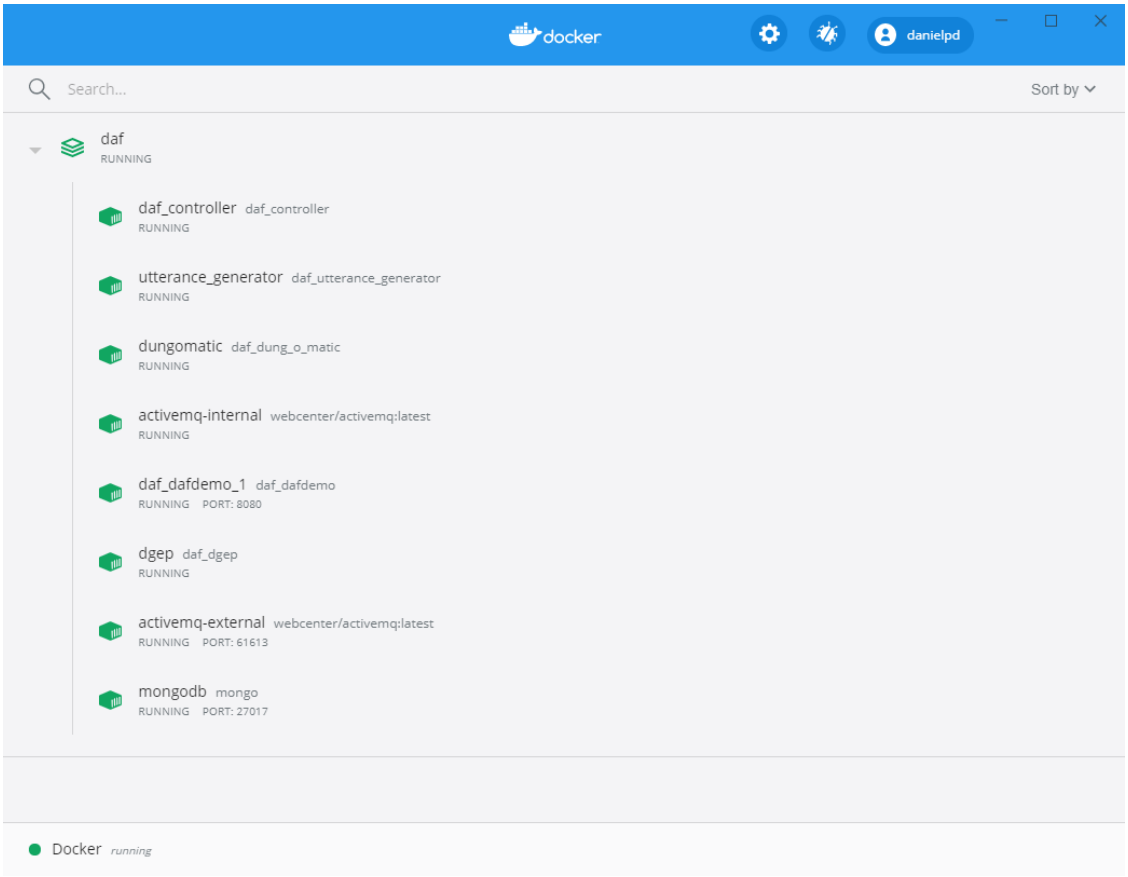


Figure 9: After running the Dialogue and Argumentation Framework from the Docker Dashboard.

Run ASAP Agent Manager

On Windows: double click the `{demonstrator}\Launchers-\ASAP_Superior_Couch_Start_NoAndroid.bat` from your File Explorer. This opens a command line shell. Wait until you see the message `Waiting for AgentSpec...`.

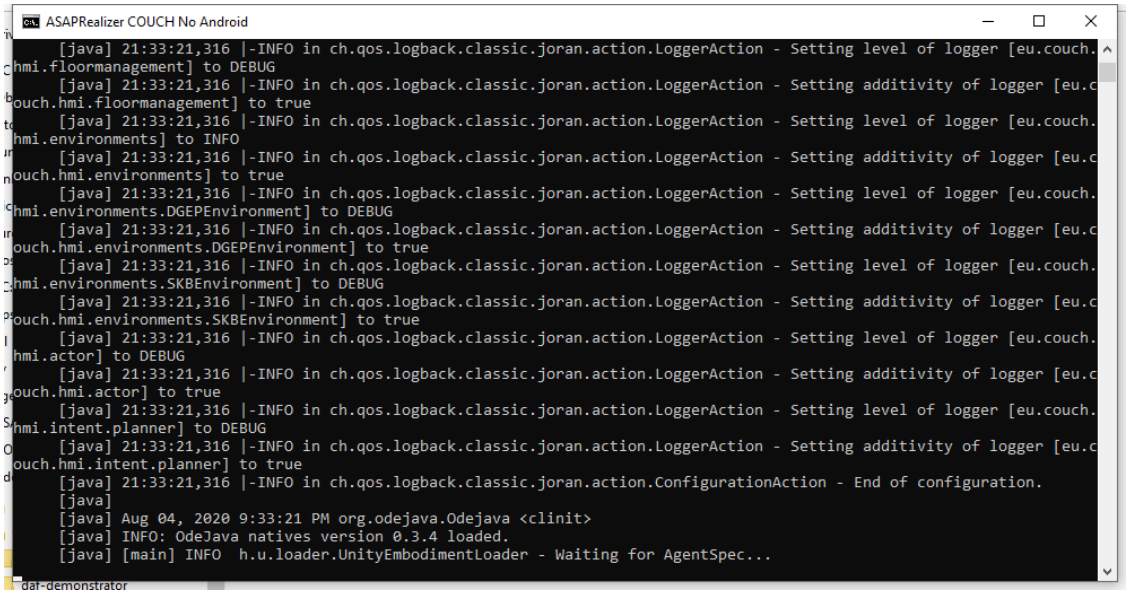


Figure 10: Running the ASAP Agent Manager.

Run Greta

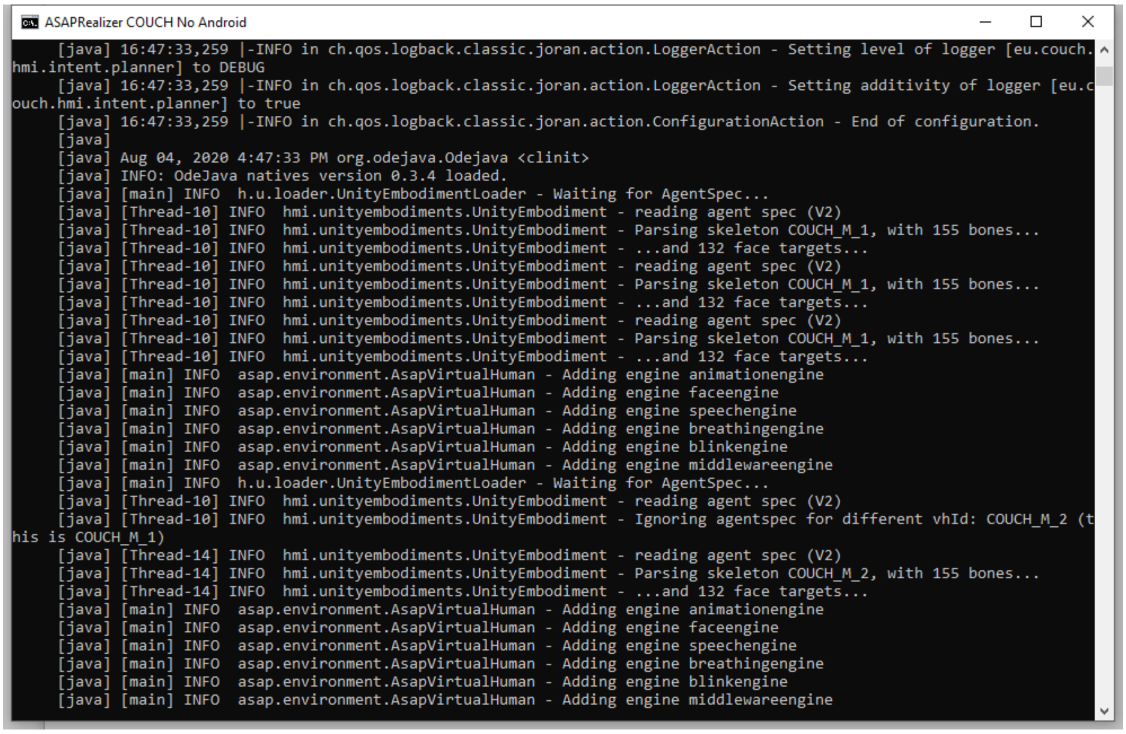
Open a command line shell, go to `{demonstrator}\greta\bin` and type the command `java -jar Modular.jar`. The Greta user interface window will open. From its menus, select File > Open and go to `{demonstrator}\greta\bin\Configurations\GretaUnity\Projects\Council of Coaches`, and select `Council of Coaches - TechnicalDemonstrator.xml`.

Run the Unity scene

Open the `AgentsUnitedDemo` project in Unity and open the `MainScene` scene. Press the Play button (usually at the top). You may be asked for firewall access. The agents briefly appear overlapping in the center of the table.

Figure 11: Result of running the GRETA Unity scene.

ASAP and Unity will now automatically create a connection and exchange details about the embodiment of the agents, as shown in the ASAP console window.




```
     [java] 16:47:33,259 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction - Setting level of logger [eu.couch.
hmi.intent.planner] to DEBUG
     [java] 16:47:33,259 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction - Setting additivity of logger [eu.c
ouch.hmi.intent.planner] to true
     [java] 16:47:33,259 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - End of configuration.
     [java]
     [java] Aug 04, 2020 4:47:33 PM org.odejava.Odejava <clinit>
     [java] INFO: OdeJava natives version 0.3.4 loaded.
     [java] [main] INFO  h.u.loader.UnityEmbodimentLoader - Waiting for AgentSpec...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - reading agent spec (V2)
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - Parsing skeleton COUCH_M_1, with 155 bones...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - ...and 132 face targets...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - reading agent spec (V2)
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - Parsing skeleton COUCH_M_1, with 155 bones...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - ...and 132 face targets...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - reading agent spec (V2)
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - Parsing skeleton COUCH_M_1, with 155 bones...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - ...and 132 face targets...
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine animationengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine faceengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine speechengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine breathingengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine blinkengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine middlewareengine
     [java] [main] INFO  h.u.loader.UnityEmbodimentLoader - Waiting for AgentSpec...
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - reading agent spec (V2)
     [java] [Thread-10] INFO  hmi.unityembodiments.UnityEmbodiment - Ignoring agentspec for different vhId: COUCH_M_2 (t
his is COUCH_M_1)
     [java] [Thread-14] INFO  hmi.unityembodiments.UnityEmbodiment - reading agent spec (V2)
     [java] [Thread-14] INFO  hmi.unityembodiments.UnityEmbodiment - Parsing skeleton COUCH_M_2, with 155 bones...
     [java] [Thread-14] INFO  hmi.unityembodiments.UnityEmbodiment - ...and 132 face targets...
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine animationengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine faceengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine speechengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine breathingengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine blinkengine
     [java] [main] INFO  asap.environment.AsapVirtualHuman - Adding engine middlewareengine
```

Figure 12: Information from the ASAP Console Window.

The agents in the Unity scene will reposition behind the table on the chairs. Note that the agents are now still standing and are overlapping with the chairs, this is normal.

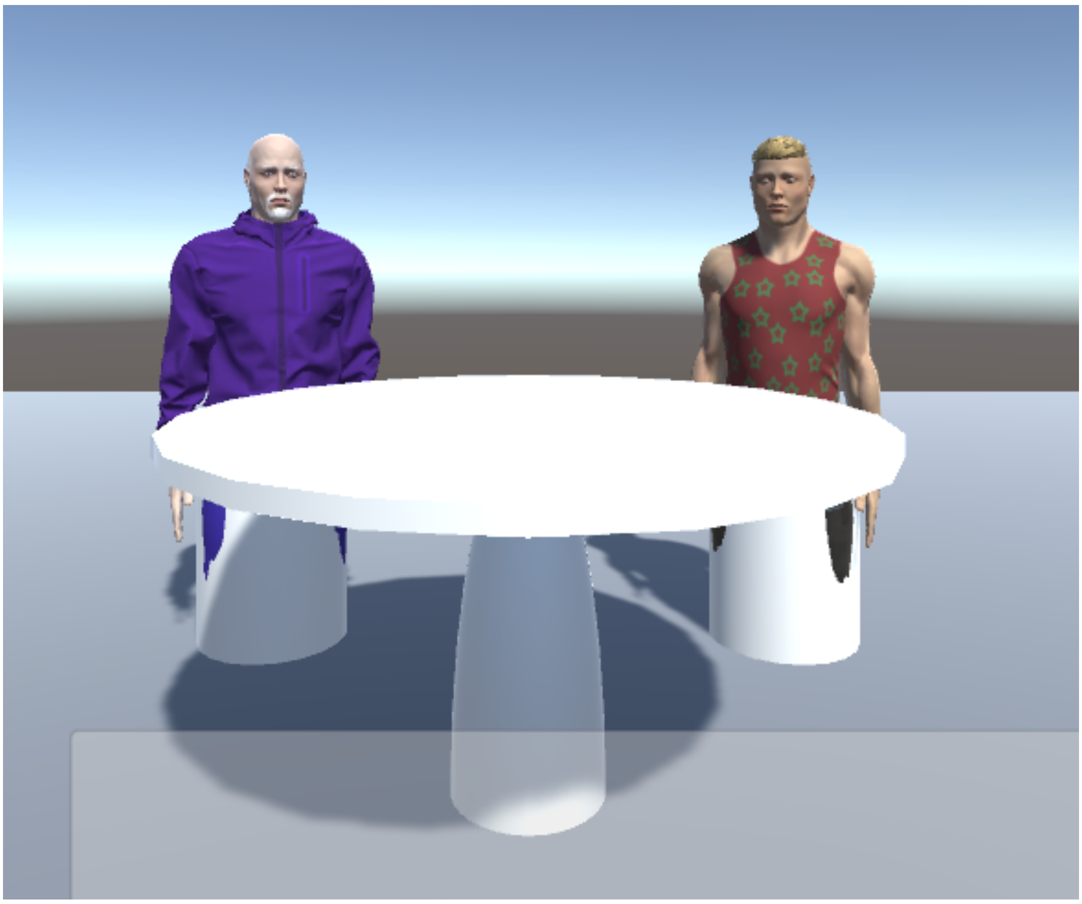


Figure 13: Result of the GRETA Unity Scene after re-configuration.

Run the Intent Planner

On Windows: double click the `{demonstrator}\Launchers\-Flipper_Superior_Couch_Start.bat` from your File Explorer. A small login window opens. The default username and password should work for connecting to the default **Wool Web Service** used in the demonstrator. If you are hosting your own service you will need to enter different login information.

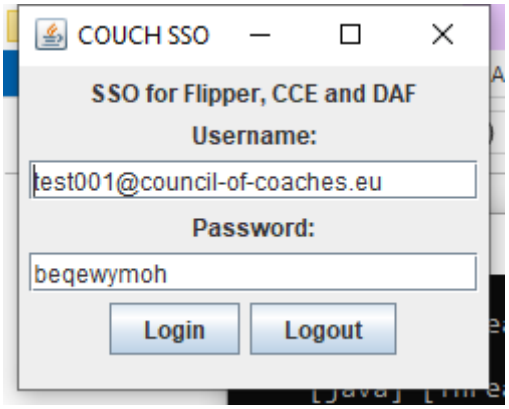


Figure 14: Single-Sign-On (SSO) widget for the Intent Planner.

Run the demo

Click the `Login` button. You are now logged in to the Wool Web Service, and your authentication key is automatically shared with the Topic Selection Engine and the Dialogue and Argumentation Framework modules.

After logging in, the demonstrator dialogue is automatically initiated. The coaches in Unity will start by saying "Hi", and will sit down on their chair. An overlay in the Unity scene will display the moves available to the user, from which you can choose how to proceed.

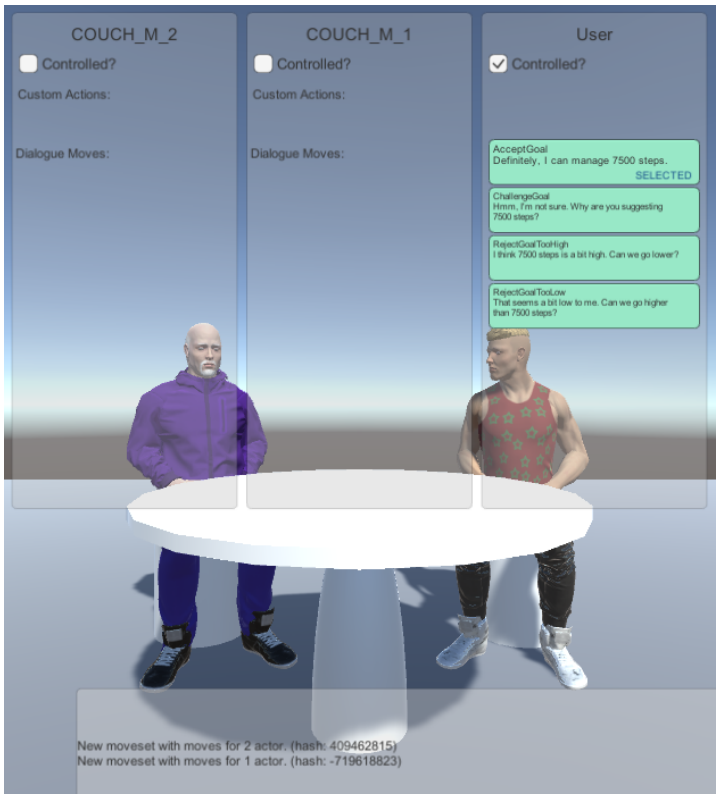


Figure 15: Final results of running the demonstrator.

The DAF console will now output information about the ongoing state of the dialogue.

Figure 16: Output of the Dialogue and Argumentation Framework console.

To restart the dialog, you need to restart only the Conversational Intent Planner (press `ctrl+c` in the console window, then `y` to confirm, then run the `Flipper_Superior_Couch_Start.bat` file again).

Stopping

1. Unity: press the Play button again (and close the Unity editor).
2. Command line windows: press `ctrl+c` and then `y` to confirm.
3. DAF: In addition to ctrl+c in its command line window, wait for Docker containers to quit, then type the command `docker-compose down`. (you can now also stop Docker).

3.4.5 Settings

3.4.5.1 Enabling user input

To change the demo between 'auto play', where the agents perform their own conversation without user input, or user input, where the user can play the role of patient (i.e. activate the UI):

- In the Flipper template file: `{demonstrator}\intent-planner\resource-\couchtemplates\DialogueLoader.xml`
- Change in the information state the variable "uidefaults" and update the identifier to set the agent that should be controlled through the interface `"uidefaults": {"actors": [{ "identifier":"User", "controlledBy": ["unityTest", "tablet"] }]}`
- Restart Conversational Intent Planner

3.4.5.2 Changing the role of the user

The role of the user can be changed. The user takes the role of one of the characters in the dialogue. Currently it is set up as the patient. To change the role of the user (i.e. the moves of the agent that the UI will show):

- Open the file: `{demonstrator}\intent-planner\resource-\couchtemplates\DialogueLoader.xml`
- Alter the `dialogueActors` variable to change roles assigned to any of the agents or the user.

```
"dialogueActors" : [
        { "bml_name":"COUCH_CAMILLE", "engine":"greta", "role":"Cami
lle",    "identifier":"COUCH_CAMILLE", "dialogueActorClass":"UIContr
ollableActor", "priority": 0.5 },
        { "bml_name":"COUCH_M_1", "engine":"ASAP", "role":"Marc",
"identifier":"COUCH_M_1", "dialogueActorClass":"UIControllableActor"
, "priority": 0.0 },
        { "bml_name":"COUCH_M_2", "engine":"ASAP", "role":"Ben",
"identifier":"COUCH_M_2", "dialogueActorClass":"UIControllableActor"
, "priority": 0.5 },
        { "bml_name":"COUCH_F_1", "engine":"ASAP", "role":"Sarah",
"identifier":"COUCH_F_1", "dialogueActorClass":"UIControllableActor"
, "priority": 0.5 },
        { "bml_name":"COUCH_M_Android_1", "engine":"ASAP", "role":"G
ordon",     "identifier":"COUCH_M_Android_1", "dialogueActorClass":"
UIControllableActor", "priority": 0.5 },
        { "bml_name":"",          "engine":"ASAP", "role":"User",
"identifier":"User",       "dialogueActorClass":"UIControllableActor
", "priority": 1.0 }
    ]
```

- Finally, restart the Intent Planner.

3.4.6 Latest fixes and troubleshooting

- By default, the Demonstrator requires Windows TTS US voices: Mark, David, Zira. install them from Windows TTS Settings and check out how to set them up with the instructions on the HMI Wiki (see https://github.com/hmi-utwente/HmiASAPWiki/wiki/MS-API-Voices). If you do not have a particular voice installed, the agents should use the default selected voice in "Windows Settings -> Time & Language -> Speech -> Voices" settings instead.
- To test the Windows voices: What you can try is run `ASAP_Superior_Couch_Start_NoAndroid.bat` from the Launchers folder and the Unity scene (do not run Conversational Intent Planner!). Once these are connected you run `BmlWindow_ASAP_Start.bat` which opens a window where you can input BML and send it to ASAP. If all is well the agent should talk. If not, try a couple more times and check the console windows for any errors or warnings. For example, the following BML will make one of the agents speak "Hello there!":

```
<bml id="sp1"  xmlns="http://www.bml-initiative.org/bml/bml-1.0" cha
racterId="COUCH_M_1">
              <speech start="0"  id = "speech1">
                <text>Hello there!</text>
              </speech>
</bml>
```

- If you had installed the DAF module before and are having problems with empty moves, you may have to clear the Move cache with: `docker exec -it mongodb mongo couch_content --eval "db.move_cache.remove({})"`
- If you want to completely delete and rebuild your DAF docker volumes and containers from scratch (this will take a while):
  1. docker-compose down --volumes
  2. docker system prune --volumes
  3. docker-compose build --no-cache
  4. docker-compose up

3.5 Dialogue GUI

This is a web-based HTML & JavaScript GUI frontend for interacting with the agents. It offers the user an intuitive interface for making decisions in the dialogue. Can be used in addition to, or instead of, the Unity GUI overlay. This project listens to ActiveMQ topics for any user-actions and generates the appropriate buttons. Has been tested in Firefox.

3.5.1 Installation

Clone this repo. Edit the config.js to subscribe to the correct ActiveMQ topics.

3.5.2 Running

Run the whole Agents United system. Open the index.html in a browser. When the user has available moves the interface will show buttons accordingly.

3.6 Documentation

This repository is a collection of screenshots and other media that is used throughout the other Agents United repositories. This is listed here for the sake of completeness.

3.7 Intent Planner

This Conversational Intent Planner (CIP) module is responsible for generating appropriate conversational intents for the group of agents in the council. It sits between the Topic Selection Engine (TSE) & Dialogue and Argumentation Framework (DAF) and the behaviour realizers ASAP & GRETA. It translates high-level dialogue moves from DAF to more low-level (group) behaviours for individual agents. Additionally, it is the main point where user moves are received and forwarded for processing. While the user and agents progress through a dialogue, it also stores certain information in the Shared Knowledge Base (SKB)

The core of this module is built upon Flipper2.0. Flipper is a rule-based dialogue engine, consisting of a collection of "templates". Each individual template specifies effects when certain preconditions are true. Flipper uses a hierarchical internal data structure (JSON) to store an information state. Preconditions are checked against the information state, and effects update the information state. It supports embedded JavaScript for light scripting and data manipulation. Heavy data processing and communications with external modules is handed off to Java objects. Within the CIP module we define several loosely-connected environments that focus on a specific task.

Conceptually, these environments are responsible for the following:

- **Talking with SKB** - Handles user authentication and storing/retrieving certain variables.
- **Talking with Topic Selection** - The CIP is the main starting point for kicking off a dialogue with a user. It asks the TSE for a dialogue that is suitable for the current user at the current time.
- **Talking with DAF** - Loads the dialogue in DAF, making sure that the various actors are all represented by an agent or the user. Then, a series of move selections is orchestrated while actors go through the dialogue.
- **Controlling the user interface** - Moves received from DAF are presented to the user/woz who can make a final selection of which move to execute. The Unity scene contains an overlay interface for each of the actors, showing the various available moves at that point in time. A user or WoZ may select a move, which is then forwarded to DAF, as described above.
- **Generating conversational intents** - Once a user/WoZ has selected a move, BML for speech and gestures is generated for each of the affected ASAP and GRETA agents. The CIP offers basic floor management and nonverbal saliency gaze behaviour.
- **ASAP** - Although ASAP is maintained in a separate repository, we include some convenient starter classes and configuration in the CIP module.

For the communication with external modules we support a variety of common middlewares through a Middleware Abstraction Layer. This allows us to easily (re)configure our module to use any combination of specific middlewares, without changing the underlying code that uses these middlewares.

We use the following specific middlewares:

- ActiveMQ for DAF, the user interface, ASAP, and GRETA
- HTTP (POST/GET) requests for the CCE and SKB (RESTful APIs)
- UDP for streaming audio and joint rotations for the various agents from ASAP to the Unity scene

3.7.1 Build

To build this module you need Ant and the HMI Build Tool, available in the **hmibuild repository**. It must be placed in the parent folder, next to the *intent-planner* folder. Then run the commands `ant clean`, `ant resolve` and `ant compile`. Take a look at how it was done in the documentation for the demonstrator repository (see Section 3.4).

3.7.1.1 Optional: use Eclipse to develop, compile and run the CIP

Of course, you are free to use which ever editor you like for editing and extending the CIP module. Compiling and running the project can always be done through the provided ant commands.

If you prefer to use the Eclipse IDE, we have additional integration options in place, which enable you to compile and build directly within the IDE.

- Download and install the latest Python 2.7. Also add it in your PATH environment variable. Make sure that the `python` command in your commandline/terminal refers to the python 2.7 version, **NOT python 3** (check with `python --version`, this should print something like `Python 2.7.18`).
- Download and install a recent version of Eclipse for Java developers. Make sure that the version you download matches with your installed java SDK (32bit or 64bit).
- In a commandline window, navigate to the main intent-planner folder and run `ant eclipseproject`. This generates the necessary Eclipse project files, making sure that all dependencies are set correctly.
- Start your Eclipse, and select `File->Import` from the main menu. This opens an Import popup window. Select `General->Existing Projects into Workspace` and press `Next`. Press `Select root directory` and click on `Browse...` to select the main `intent-planner` directory. The `IntentPlanner` project should now show up in the `Projects:` area. Make sure the project is selected. (None of the other options on this page should be selected.) Press the `Finish` button to import the project in Eclipse.
- The Java source files are now automatically compiled as you edit them. The flipper templates are interpreted at run-time. You can launch the various components of this module from the `src/eu.couch.hmi.starters` package. `AsapCouchStarter` starts the ASAP realizer, and `FlipperStarter` starts the main CIP flipper instance.

Whenever you make modifications to the dependencies in the `ivy.xml` file, you will need to re-run the `ant resolve`, `ant compile`, and `ant eclipseproject` commands. You can then right-click on the project in Eclipse and select `Refresh` to import the latest dependencies.

3.7.2 Configuring the system

3.7.2.1 ASAP agents

Within the CIP module we include some helpful configuration and launchers for running the ASAPRealizer. The main configuration file is `resource/couchlaunch.json`. Here, you configure which agents (charId) should be loaded by ASAP, according to which specification files (spec).

The actual configuration for each agent is located in the spec files. By default, `multiAgentSpecs/uma/UMA1.xml` and `multiAgentSpecs/uma/UMA3.xml` are loaded for agent COUCH_M_1 and COUCH_M_2, respectively. These files specify exacyly the properties of each agent. Most options should be left at default values. Things you may want to change:

Voice of the agent: By default, the ASAP agents are configured to use Windows/Microsoft MSAPI voices. You can modify the property `<Voice factory="WAV_TTS" voicename="Microsoft Mark" />` to select any of the installed MSAPI voices on your Windows machine.

Alternatively, you may switch to MaryTTS. This may be useful if you are using OSX or Linux. In the agent spec file, you should modify the loaders `ttsbinding` and `speechengine` to load MaryTTS instead:

```
<Loader id="ttsbinding" loader="asap.marytts5binding.loader.MaryTTSBindingLoader">
    <PhonemeToVisemeMapping resources="Humanoids/shared/phoneme2viseme/" filename="sampaen2disney.xml"/>
</Loader>
<Loader id="speechengine" loader="asap.speechengine.loader.SpeechEngineLoader" requiredloaders="facelipsync,ttsbinding">
    <Voice factory="WAV_TTS" voicename="dfki-spike"/>
</Loader>
```

Then, add the selected voice as a dependency in the `ivy.xml` file (For example: `<dependency org="marytts" name="voice-dfki-spike" rev="latest.release" />`) and run `ant resolve` and `ant compile` in a terminal at the root of the `intent-planner` directory. Note that ASAP uses an internal embedded MaryTTS instance, which is separate from the MaryTTS standalone server required by Greta.

Gesture bindings: More advanced users may want to take a look at the gesturebinding, which specifies how generic BML is mapped onto specific supported behaviours/gestures/animations of the agent: `<GestureBinding basedir="" resources="" filename="Humanoids/uma/gesturebinding/gesturebinding_borg.xml"/>`

Middleware communication channels: BML commands and BML feedback for each ASAP agent are sent/received through a shared "BML port", which is configured to use a middleware. The parameters for the middleware are configured in `resource/multiAgentSpecs/shared_port.xml`. By default, these are configured as ActiveMQ topics.

A different shared middleware is used to stream agents' joint rotations to the Unity scene. These are configured in `resource/multiAgentSpecs/shared_middleware.xml`. Since streaming this data should happen realtime and is relatively high-volume, we use UDP by default.

3.7.2.2 Flipper and the Conversational Intent Planner

The main configuration file for Flipper is `resource/couchflipper.properties`, which specifies the collection of template files to load.

The actual templates are located in `resource/couchtemplates`. Settings for the various environments are configured in `Environment.xml`, in the information state variable `"environmentSpec": {"environments": [.....]}`. For each environment you may specify `requiredLoaders` (i.e. other environments it depends on) and additional `params` that are specific to the environment. Environments that communicate with other external modules require a middleware specification as a parameter.

authEnv: The authentication environment is responsible for authenticating the user with the Wool Web Service, and subsequently sharing the authentication token with other modules to allow a single-sign-on (SSO) entry point to the system. It requires several middleware instances:

- `AuthServiceMiddleware`: The service that does the authentication and returns an authentication token used for SSO.
- `SSOMiddleware`: The middleware on which we publish the SSO token. Any external module may listen to this middleware for an authentication token if they wish to automatically log in the user.
- `SSOCCEMiddleware`: A special middleware for transmitting the SSO token to the topic selection engine, which cannot listen to the regular `SSOMiddleware`.

Additionally, a default `username` and `password` may be configured, which will be pre-filled in the login GUI.

bml: The environment for sending BML to the ASAP realizer and receiving feedback on the execution of the BML through the configured `middleware`. If `publishBmlFeedback` is set to true, feedback is also published to Flipper, enabling it to be used in template preconditions.

fml: The environment for sending FML and BML to the Greta realizer and receiving feedback through the specified `middleware` and `middlewarebml`. Greta uses a unique middleware channel for each agent. For example, Greta's `COUCH_CAMILLE` agent will listen to FML requests on topic `COUCH/FML/REQUESTS/COUCH_CAMILLE`. `characterIds` lists the agent IDs that are controlled by Greta, these IDs will be appended to the `iTopicPrefix` configured in each middleware. If `publishFmlFeedback` is set to true, feedback is also published to Flipper, enabling it to be used in template preconditions.

asapIntent and gretaIntent: These environments translate a high-level move from DAF into behavioural intents and actual specific BML and FML for each respective realizer. A move may specify a template file with BML or FML, located in `bmlTemplateDir` or `fmlTemplateDir`. If no such template file is specified, or it can not be found, default BML and FML are generated dynamically.

skb: The SKB environment connects to the Wool Web Service for storing and retrieving variables through the `set-variables` and `get-variables` middleware, respectively.

dgep: The DGEP environment connects to DAF and is responsible for initiating dialogues and handling the flow of moves during an interaction. The `dgepCtrlMiddleware` is used for initiating the dialogue with actors. The `dgepMovesMiddleware` is used for selecting specific moves and retrieving subsequent available moves throughout a running dialogue.

simplefilter and flipperMoveProxy: These are basic environments for filtering incoming moves. In the demonstrator, these default environments don't do any filtering. More advanced implementations may want to filter incoming moves based on the user's preferences, the interaction history, or other considerations.

cce: The cce engine talks through the `get-new-topic` middleware to the topic selection engine, to retrieve a high-level topic for the logged in user. Once a topic is retrieved, it is initated in DAF. The retrieved topic may contain additional parameters that are passed on to the DAF.

ui: As a dialogue progresses, the UI environment listens in on the incoming moves. Through the configured `middleware` it instructs any (graphical) user interfaces to display the moves to the user. It is up to the implementing user interface how these choices are represented exactly. For example, in the basic demonstrator, the Unity generates an overlay with buttons. Additionally, we have several mobile interfaces available, which emulate WhatsApp or chatbox style interactions.

floormanagement: When there are several agents that want to make a move at the same time, the floormanagement environment takes care of selecting which agent may take the floor. There are several built-in floor management styles available, which can be configured in the `style` field:

- **FCFS**: a very basic first-come-first-serve floor manager. The floor will be granted to which ever agent requests the floor first.
- **RANDOM**: a basic floor manager which waits for a while to collect moves from agents (autonomous, wizard-controlled, and user-controlled) and then hands the floor to one at random.
- **GBM**: a more advanced external "Group Behaviour Module", implemented as part of Greta. This floor manager communicates through the configured `middleware`. Currently under construction. The GBM will offer more intelligent floor management and also (nonverbal) turntaking behaviours.

dialogueloader: Ties together various other environments for initiating a dialogue, handling moves, generating behaviours. Has no configuration options.

sse: The social saliency environment listens for BML and FML commands and feedback, and generates basic accompanying nonverbal social saliency behaviours. Currently it generates gaze, more advanced implementations may also generate gestures and backchanneling, for example.

3.7.3 License

This module is licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0).

3.8 UnityProject

There are several different Unity projects, each with their own purpose:

- **COUCHUnityProject & AgentsUnitedDemo**: The main projects for displaying agents from ASAP and GRETA in one scene. These also contains a basic interface overlay for controlling moves for each of the agents and the user.
- **CharacterCreatorNew**: Can be used to create and generate new UMA characters, which can then be imported as ASAP agents in other projects.
- **OculusQuest**: A very basic proof-of-concept project that is configured to run natively on the Oculus Quest VR headset using only ASAP agents.
- **WizardInterface**: A Wizard of Oz interface that is configured to run on an Android tablet or phone.

3.8.1 Build

The *AgentsUnitedDemo* project uses only free/open source assets and is configured out of the box with ASAP and GRETA agents.

The projects *COUCHUnityProject* and *OculusQuest* use some commercial 3rd party assets that cannot be made available in this repository. Please purchase and/or download these assets from the Unity Asset store and place them in `{project}\Assets\Borg\3rdParty`:

- Devotid Folding Table and Chair
- DigitalKonstrukt Prototyping Pack
- o3n Male and Female UMA Races
- o3n Modern Clothing Pack for Male
- o3n Modern Clothing Pack for Female
- RecompileDisabler
- RootMotion Final IK

3.8.2 Run

1. Start Unity and open AgentsUnitedDemo, then open scene Assets/AgentsUnited/Scenes/MainScene.
2. You should see a room with 2 basic stools and a table. There should be 2 male grey agents on top of the stools (placeholders for ASAP agents) and 2 standing female agents (GRETA agents).
3. Follow the steps in the demonstrator readme to initialise and run the other modules that actually control the behaviour of the agents.

3.8.3 Documentation

The following GameObjects/Components do most of the work in the scene.

3.8.3.1 ASAPToolkit

The ASAPToolkit/ASAPToolkitManager connects to an external ASAPRealizer module, which controls the various ASAP agents movements and speech (see AudioStreaming/AudioStreamingReceiver). At initialisation time, the ASAPRealizer module requests the skeleton joint/bone map from the agents in the Unity scene. While running, it then streams joint rotations for the full skeleton to the Unity scene to animate each of the agents. Network connection parameters are configured in the various xxxMiddleware component scripts.

WorldObject components can be added to any GameObject to automatically make them available in BML as gaze/point targets.

3.8.3.2 UMA_DCS

The UMA agents are used by ASAP. UMA uses recipes to define how an agent looks and what it is wearing (see UMA documentation for more details). When adding new recipes (or other library assets) make sure to update the global UMA library: in toolbar: UMA->Global Library Window->add individual assets or rebuild index

The UMA agents in the scene are COUCH_M_1 and COUCH_M_2, which are DynamicCharacterAvatars. The BasicCharacter component initialises each agent and adds it to the ASAPToolkitManager through its unique AgentID. The AgentID is used to send BML to specific agents. Our DynamicUMASkeleton component is responsible for mapping ASAPRealizer's joint representation onto an UMA agent's bones.

3.8.3.3 Canvas

The UIMiddlewareMoves displays a basic user interface overlay for controlling agent and user moves. Is generated dynamically at runtime based on the characters currently active in the ongoing dialogue. Toggle between user-controlled moves and automatic move selection. Buttons will typically display the text that will be spoken by the agent/user. May also be used for Wizard of Oz (WoZ) purposes.

3.8.3.4 Camille and Laura

These agents connect to a running GRETA FML realizer module, which controls their animations and speech.

3.8.4 Troubleshooting

3.8.4.1 Using GIT with UNITY

We use GIT with Unity, but there may be merge conflicts if you push and pull your project. The following workflow seems to be ok to solve the conflicts:

1. First, install DiffMerge. This is the graphical tool that you will use to merge conflicts in Unity asset and scene files.
2. Second, add `UnityYAMLMerge.exe` to your windows PATH variable. This tool will automatically try to merge different versions of the assets. It should be installed in your Unity folder, somewhere like: `C:\Program Files\Unity\Hub\Editor\2017.4.32f1\Editor\Data\Tools\`
3. Then, configure GIT to use UnityYAMLMerge. Add the following to your `~\.gitconfig` file:

```
    [merge]
        tool = unityyamlmerge
    [mergetool "unityyamlmerge"]
        trustExitCode = false
        cmd = 'UnityYAMLMerge.exe' merge -p "$BASE" "$REMOTE" "$LOCA
L" "$MERGED"
```

If there are conflicts after a commit, pull or stash pop or whatever, GIT will tell you about them. You can then enter command `git mergetool` which will call the UnityYAMLMerge automatically. If there are still conflicts that then can not be resolved automagically, it will launch the graphical editor DiffMerge. You make your merges here by hand by picking changes from your file and the remote file (left is local, middle is the resolved new file, right is remote file).

3.8.5 License

These projects are licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0).

3.9 universAAL

The projects in the **universaal** repository are applications used to connect the Agents United platform to the universAAL IoT Platform. This allows Agents United to incorporate sensor data from devices

connected in universAAL. Right now we only connect to data from weight-scales and blood pressure sensors.

- **uaal.hbaf**: This is a Java Maven project of an universAAL application that acts as a Relay App: It is run as a bundle in a Karaf OSGi instance of universAAL. It receives any data coming from weight-scales and blood pressure sensors connected to universAAL, and then delivers this data to Agents United HBAF module.
- **uaal.mobile**: This is an Android app. It connects to Continua-certified weight-scales and blood pressure sensors and provides a user interface to take their measurements. Then it sends this data to the Karaf OSGi instance of universAAL where uaal.hbaf is running.
- **uaal.coaching**: This is a Java Maven project of an universAAL application that allows other universAAL apps to start coaching sessions in Agents United. It is run as a bundle in a Karaf OSGi instance of universAAL. It publishes a universAAL service based on the coaching ontology (see below) to start coaching sessions, by sending the command to Agents United.
- **uaal.coaching.ont**: This is a Java Maven project of an universAAL ontology representing the coaching domain. It is added as a bundle in a Karaf OSGi instance of universAAL.

3.9.1 Build

The **uaal.hbaf**, **uaal.coaching** and **uaal.coaching.ont** projects are Java Maven projects that produce an OSGi bundle. You need to have Maven installed and then execute mvn install in the folder where the pom.xml is.

The **uaal.mobile** project is an Android app. You can use Android sdk or the Android Studio IDE to build the .apk.

3.9.2 Run

3.9.2.1 Relay app

This is an universAAL bundle, you need to run it in a Karaf OSGi instance of universAAL. Then you will need to install the universAAL REST API there so that the Mobile App can send the data there.

1. Go to universAAL's Github and clone or download the Karaf distro. The latest version, which is the one compatible with our code, is not yet released, so you will have to build the distro. Follow the instructions in https://github.com/universAAL/distro.karaf. (essentially, clone/download, then mvn install)
2. Before you run the Karaf distro (or afterwards, as long as you restart), go to {your karaf folder}\etc and edit system.properties.
    - Add this property org.universAAL.ri.rest.manager.serv.host=http://192.168.1.1:9000/ and set the IP to your host's IP. This is where the universAAL REST API will be published. Make sure it's an IP the Mobile App will be able to access.
    - Add this property eu.councilofcoaches.uaal.hbaf.url and set it to the base URL where your Agents United HBAF server is running.
3. Run the universAAL Karaf distro by running the {your karaf folder}\bin\karaf script. Once you see 5 "GMS" messages in the console, it is up and running.
4. Install the universAAL REST API and the needed ontologies with the following commands in the Karaf console:
    - feature:install uAAL-Ont.Health.Measurement
    - feature:install uAAL-Ont.personalhealthdevice
    - feature:install uAAL-Ont.Profile
    - feature:install uAAL-Ont.Security

- feature:install uAAL-Ont.Cryptographic
- feature:install uAAL-RI.RESTAPI

5. Place the following libraries (links to their Maven repository entires below) .jar files manually in {your karaf folder}\deploy or through the Karaf console with the command install mvn:{groupId}/{artifactId}/{version}:
   - https://mvnrepository.com/artifact/commons-codec/commons-codec/1.11
   - https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core/2.9.8
   - https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations/2.9.0
   - https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.9.8
   - https://mvnrepository.com/artifact/com.auth0/java-jwt/3.8.0
6. Place the uaal.hbaf Relay App .jar file (you will find it in its /target folder after you build it) and place it in the {your karaf folder}\deploy folder.
7. The Relay App should now be ready to relay sensor data that arrives from the Mobile App through the REST API to the Agents United HBAF. You can check that all bundles are "Active" with the Karaf command list.

3.9.2.2 Mobile app

This is an Android App. You can use Android Studio or the Android sdk tools to build the .apk and then push it and install it in a mobile phone. The app requires a phone with Android 5.0 and Bluetooth Low Energy. It has been tested with A&D Medical devices UA-651 and UC-352. The devices must be paired *before* launching app. Once inside the app, follow these steps:

1. Open the Connection settings throgh the upper-right corner menu (If you are asked a developer PIN, it is 8225).
2. Find the "Remote connecion URL" setting and set it to your universAAL REST API address (The one you set up in the previous instructions, appending /uaal at the end).
3. Enter your credentials, which should have been created in advance and registered in HBAF.
4. Press Login, and wait until the app validates, connects and starts. If it does not connect after a while, force close the app, and start again.
5. Once the app connects you will see a button for each type of device to take the measurement. Press it, then take the measurement, and wait for it to appear in the screen. It will be immediately sent to the universAAL REST API (You should see a message pop up in the Karaf console).

3.9.3 UniversAAL Agents United Coaching App integration

Coming soon.

3.9.4 Troubleshooting

- If you get an error message in the app after taking a measurement about not connecting to the server, try logout (upper right menu) and login again.

3.9.5 License

These modules are licensed under the GNU Lesser General Public License v3.0 (LGPL 3.0).

4 Documentation for Individual Platform Components

The Agents United Platform documentation described in Section 3 focuses on the "entry point" into the overall Agents United Platform, which is an integration of a number of platforms that can also be used in a stand-alone fashion. For example, the GRETA platform, developed and maintained by Sorbonne University is an embodied agent platform that has been integrated with, and can be used as part of the Agents United Platform, but has existed as a stand-alone platform for 3D virtual agent embodiment for many years prior. The same holds for the ASAP platform which was developed over the past years by the Human Media Interaction group of the University of Twente, as well as the Dialogue and Argumentation Framework developed and maintained by the University of Dundee.

It is thus out of the scope of the Council of Coaches project to present all detailed documentation for those platforms that existed as background of the project, outside of those functionalities and integrations describe in Section 3 above, and additional documentation that was generated within the scope of the project.

Below, we document for those individual platform components what was generated as training material within the scope of the Council of Coaches project.

4.1 WOOL Platform Documentation

The WOOL Platform has been developed from the ground up within the Council of Coaches project, as the underlying platform for the project's Functional Demonstrator, as well as to drive scripted dialogues within the Agents United platform. As such, WOOL can be seen as a second technical open source platform released by the project, and we will focus on this platform's release and documentation in this section.

Up to date documentation for the WOOL Platform is developed for and maintained at the specific WOOL website set up at www.woolplatform.eu. The contents below reflect a snapshot of that website as of August 2020.

4.1.1 What is WOOL?

At its core, WOOL is a language definition that can be used to write dialogue. The WOOL platform consists of (1) the dialogue language definition, (2) a powerful editor for creating dialogue scripts, and (3) a set of tools to execute these dialogues within your application of choice.

At the moment, we have released the language definition (for which the documentation is described below in Section 4.1.2), the editor that can be either downloaded from the WOOL GitHub page (WOOLPlatform.EU, 2020), or used online through the web-based editor (WOOLPlatform.EU, 2020), and a parser/executor library developed both for JavaScript and for Java that can also be downloaded from GitHub.

The WOOL language is based on existing open source frameworks such as Yarn (Secret Lab, 2020) and Twine (Twine, 2020). Whereas Twine is designed for interactive storytelling, and Yarn has its roots in video game development, WOOL was developed in the context of designing conversational agents for health promotion.

The WOOL language was specifically created in the context of the Council of Coaches project.

4.1.2 The WOOL Language Documentation

The basis of WOOL is a scripting language specifically defined to allow authors to write branching dialogue. Any individual or organisation that wished to adopt WOOL in their own applications, should first familiarize themselves with this scripting language. The language itself has shaped up during the course of the Council of Coaches project in order to accommodate new features as they were being implemented in the project's demonstrators. At the current point in time, the basic language is considered stable, although many additional features are still one the "wish list" and are being designed. A full up-to-date documentation of the language is available on the WOOL Platform website at

www.woolplatform.eu. The current version of this material is presented in Annex A: WOOL Language Documentation.

4.1.3 WOOL Tutorials

In order to help developers on their first steps into using WOOL, a series of Tutorials are planned to be released, two of which have been completed to date, and are provided here. The first tutorial is a technical walkthrough on how to embed the WOOL Java Libraries in your own Java project. The second tutorial takes a playful approach to explaining the major WOOL Editor features, showing how you can use WOOL in a different context – of making a web-based video game.

The tutorial material is provided in the following annexes:

- Annex B: WOOL Tutorial: Setting up WOOL for your Java project
- Annex C: WOOL Editor Tutorial: How to make a standalone interactive fiction game.

4.2 ASAP Platform Documentation

In the COUCH project, the ASAP realizer is included in the Conversational Intent Planner module. Documentation for setting up and running ASAP as part of the COUCH system are in Sections 0 and 0. Documentation for configuring agents controlled by ASAP realizer is in Section 3.7.2.1.

More extensive documentation regarding ASAP can be found here:
https://github.com/ArticulatedSocialAgentsPlatform/Asap/wiki

4.3 GRETA Platform Documentation

GRETA is a real-time three-dimensional embodied conversational agent platform with human 3D models, compliant with MPEG-4 animation standard. GRETA is able to communicate using a rich palette of verbal and nonverbal behaviours. Greta can talk and at the same time also shows facial expressions, gestures, gaze, and head movements. Two standard XML languages, namely FML and BML, allow the user to define her communicative intentions and behaviours respectively based on standard SAIBA architecture. The FML and the BML can also be sent through the network by ActiveMQ. The GRETA agents are displayed either in Ogre or in Unity 3D.

GRETA is released in either LGPL v3 license or GPL v3 license. The GPL release contains some components which are not LGPL compatible. The release used for this project is the LGPL release. Both releases are available at https://github.com/isir/greta. The releases also contain the tools to create gestures and facial expressions. The platform can be used to control one or multiple agents conversing together or with a human participant. Since 2000, GRETA has been developed through many French and European projects. Further explanation about GRETA can be found at https://github.com/isir/greta/wiki.

4.4 Dialogue and Argumentation Framework Documentation

The Dialogue and Argumentation Framework has seen development before the Council of Coaches project, but has now seen a great leap in maturity. As such, a lot of efforts have been put in generating appropriate documentation for the platform to be used by interested 3rd parties. The documentation to assists users in installing the platform, running it and creating new content is provided here below.

4.4.1 Prerequisites

1. The DAF requires Docker (https://www.docker.com), including docker-compose
2. Download or clone the DAF repository from https://github.com/AgentsUnited/daf

4.4.2 Running

1. Open a command line terminal and move into the root of the repository
2. Run the command **docker-compose up**. The containers will start to build. This process may take several minutes.

3. When the DAF is ready, the following will be printed in the terminal:

   controller_1 | Dialogue and Argumentation Framework ready

4. Open a web browser and enter:

   http://localhost:8080

   The management Console will load.

4.4.3 Creating a new protocol

1. In the DAF Management Console, click "Edit protocols".
2. Click "New protocol".
3. Provide a name for your new protocol, and enter the DGDL description in the large text box.
4. Click "Save"; if there are no errors, you will see a popup "Protocol saved"; if there are errors, you will see a popup describing them; note the following:
   a. Many displayed errors can be caused by a single mistake in the DGDL.
   b. "Mismatched input" refers to a syntax error and will provide the line number and character.
   c. Fixing a "mismatched input" error can clear all other errors, so try saving again once you've identified the issue.

4.4.4 Creating content

Content for dialogues consists of three connected aspects:

1. **Coaching variables** allow temporary values to be assigned to variables for testing protocols;
2. **Argument models** are used to construct arguments on the basis of variables in the knowledge base;
3. The **Dictionary** is used to map logical argument structures to natural language utterances;
4. **Content descriptors** are used to define queries that map content to move types.

The DAF allows you to set temporary **coaching variables** for testing protocols without needing to make changes to a permanent variable store. To edit these test variables, click "Coaching variables" and change the names and/or values, or add new variables with the "Add" button. To remove a variable, delete its name.

To add and edit **argument models,** in the DAF Management Console, click "Edit content" then "Argument models". Click the name of the protocol whose models you want to edit and you will be presented with a window with three tabs, which are explained below.

Argument models use a Prolog-style representation for statements:

- *`foo(bar)`* refers to the literal string "bar";
- `foo(100)` refers to the literal integer 100;
- *`foo(Bar)`* refers to a variable Bar;
- *`foo(bar, Bar)`* refers to the literal string "bar" and the variable Bar

Coaching variables are converted into statements ready to be used as the primary source of data for the argument models, without needing to connect to the Wool Web Service (e.g. for testing purposes). For example, a coaching variable "steps" with a value "5000" will be represented as:

```
steps(5000)
```

Rules are used to define how arguments should be constructed based on the coaching variables, or other arguments. The basic structure of a rule is:

`statement[,statement,…] => statement`

As with Prolog, statements in rules can contain either atoms or variables in their arguments, or a combination of both. For example:

```
1. weather(sunny), temperature(warm) => go_to(beach);
```


- "If the weather is sunny and the temperature is warm, go to the beach"

2. `preferred_goal(Goal) => achievable_goal(Goal)`
   - "For any Goal, If Goal is preferred then Goal is achievable"
3. `achievable_goal(Goal) => set_goal(Goal)`
   - "For any Goal, if Goal is achievable, set the goal Goal"
4. `is(Food, tasy) => eat(Food);`
   - "For any Food, if Food is tasty, eat Food"

Where rules contain variables they are instantiated on the basis of the coaching variables, by chaining other rules. To use rules 2 and 3 from above, assume there is a coaching variable:

```
preferred_goal = 10000
```

which means we have the following statement:

```
preferred_goal(10000)
```

On the basis of rule 2, we obtain the following concrete argument:

```
preferred_goal(10000) => achievable_goal(10000)
```

"If 10000 is preferred, 10000 is achievable"

This now allows us to construct another argument on the basis of rule 3:

```
achievable_goal(10000) => set_goal(10000)
```

"If 10000 is achievable, set the goal 10000"

Note that the variable names are arbitrary and do not influence the chaining, e.g. rule 2 could just as easily have been:

```
preferred_goal(Foo) => achievable_goal(Foo)
```

And it still would have chained with `achievable_goal(Goal) => set_goal(Goal)`.

The format for entering rules is one rule per line, terminated with a semi-colon, i.e.:

```
a(X) => b(X);
c(Y) => d(Y);
```

Preferences are used to indicate which arguments are preferred, relative to other arguments. To continue the *goalsetting* example from above, assume a second coaching variable:

```
lowergoal = 7000
```

leading to the statement:

```
lower_goal(7000)
```

We can indicate that our "preferred goal" is more preferred than our "lower goal" (or, conversely, our "lower goal" is less preferred to our "preferred goal") in the following way:

```
Lower_goal(7000) < preferred_goal(10000)
```

This can also be represented more generally as:

```
lower_goal(Lower) < preferred_goal(Preferred)
```

Where "`Lower`" and "`Preferred`" are variables such that `Lower != Preferred`.

Preferences between statements propagate through the arguments they are a part of. Assume we have the following rules:

1. `preferred_goal(Preferred) => achievable(Preferred)`
2. `achievable(Achievable) => set_goal(Achievable)`
3. `lower_goal(Lower) => reasonable(Lower)`
4. `reasonable(Lower) => set_goal(Lower)`

the previously defined preference:

```
lower_goal(Lower) < preferred_goal(Preferred)
```

and the following coaching variables (as statements):

1. `preferred_goal(10000)`
2. `lower_Goal(7000)`

The following two arguments for set_goal are constructed:

1. `preferred_goal(10000) => achievable(10000) => set_goal(10000);`
2. `lower_goal(7000) => reasonable(7000) => set_goal(7000);`

Because all "`lower_goal`" statements are less preferred to all "`preferred_goal`" statements, argument 1 for `set_goal(10000)` is more preferred to argument 2 for `set_goal(7000)`.

Such preferences between arguments are primarily used in addressing conflict, which is expressed via *contrariness*, described in 4.2.4 below.

As with rules, preferences are expressed one per line, terminated by a semi-colon:

```
a(X) < b(Y);
c(Y) < d(Z);
```

Contrariness defines conflict between portions of an argument and is expressed as follows:

```
foo(bar)^boo(far)
```

This means that foo(bar) is in conflict with boo(far) but not the other way around; this can be loosely interpreted as foo(bar) "winning" against boo(far). To express mutual conflict, the syntax is:

```
foo(bar)-boo(far)
```

This is shorthand for:

```
foo(bar)^boo(far)
boo(far)^foo(bar)
```

As with rules, contrariness can also be expressed over variables:

```
foo(X)^boo(Y)
```

The same variable can be used in both statements:

```
foo(X)^boo(X)
```

Finally, contrariness can be expressed between the same statement, but with different variable values:

```
foo(X)-foo(Y);
```

This means that for any value of X and Y where X != Y, the "foo" statements are in conflict with each other. More concretely, to continue the goal-setting example, we can define:

```
set_goal(X)-set_goal(Y)
```

This means that any pair of arguments that propose setting different goals are in conflict with each other. If we revist the arguments built in section xxx:

1. `preferred_goal(10000) => achievable(10000) => set_goal(10000);`
2. `lower_goal(7000) => reasonable(7000) => set_goal(7000);`

we can see that set_goal(10000) is in conflict with set_goal(7000) and vice versa. In other words, we are modelling goal-setting as a decision problem: you can chose one and only one value for the goal.

Combining contrariness with preferences helps make the decision. Recall that, thanks to the preference:

```
lower_goal(Lower) < preferred_goal(Preferred)
```

argument 1 is more preferred to argument 2, which in turn means argument 1 **defeats** argument 2. This resolves the conflict: the accepted argument is set_goal(10000).

Contrariness can also help an agent adapt as a dialogue unfolds. Let us assume a new contrary:

```
rejected_goal(Rejected)^preferred_goal(Rejected)
```

In other words, for all values of “Rejected”, rejected_goal is in conflict with preferred_goal (but not the other way round).

If as a result of a dialogue, we obtain a coaching variable (expressed as a statement):

```
rejected_goal(10000)
```

Then we have a concrete contrary:

```
rejected_goal(10000)^preferred_goal(10000)
```

Since this conflict is only in one direction, we do not need a preference to resolve it: rejected_goal(10000) therefore wins, and so preferred_goal(10000) is defeated. This defeat propagates through the arguments, and thus argument 1 as a whole is defeated. This in turn renders the defeat of argument 2 ineffective and as such our new accepted argument is for set_goal(7000).

As before, the format for entering contraries as one per line, terminated by a semi-colon:

```
a(X)^b(X);
c(Y)^d(Z);
```

The **dictionary** maps the logical expressions in argument rules to natural language utterances. To edit the dictionary, click “Edit content” then “Dictionary”. The dictionary has the following hierarchy:

```
Protocol -> Statement -> Move type -> Delivery style -> Utterance
```

For example:

`goalsetting -> set_goal(X) -> propose -> authoritative ->` We should set you a goal of `$X`

When a new protocol is created, its name will automatically be added to the list; simply click the protocol you want to edit the dictionary for. Either select an existing statement, or click “+” to add a new one. Statements should map to available arguments or coaching variable statements and are expressed in the same way, e.g.

```
set_goal(X)
```

After selecting or creating a statement, you can select or create a move type. These map to the move types defined in the DGDL specification for the protocol.

After selecting or creating a move type, you will be presented with a tabular interface. This is where the actual utterances are created, mapped to delivery styles. To add a new delivery style, click the “+”. Currently, three values are supported here: “authoritative”, “socratic”, and “*” - where * represents the default style.

After creating or selecting a delivery style, you can create the utterance. Variables from the statement can be referenced by using $<name>; for instance, given a statement set_goal(Goal), you can define the utterance:

> We should set you a goal of $Goal

Which means that for a concrete statement set_goal(10000) the utterance will be:

> We should set you a goal of 10000.

Coaching variable values can also be referenced using {<name>}. To expand the above example, we can create the utterance:

> We should set you a goal of $Goal {current_goal_type}

Then, with a statement of set_goal(10000) and a coaching variable current_goal_type = “steps”, the utterance would be:

> We should set you a goal of 10000 steps

5 Webinars and Workshops

Outside those workshops held in the context of the work on Responsible Research and Innovation (see D2.7 and D2.8) and the project's final Workshop reported in D8.10, the Council of Coaches has organized a Webinar on the use of FIWARE/universAAL, described below in Section 5.1. Furthermore, two additional workshops will be held after the project's official closing date: a workshop on virtual agents, and specifically the Agents United Platform (see Section 5.2) and a workshop on evaluation methods for digital health innovations (see Section 5.3).

5.1 Webinar on FIWARE/universAAL

Early in the start of the project, the Consortium spent a good deal of research on the topic of using the FIWARE and/or universAAL platforms to integrate our software platform on. In this context, in November of 2017, the Polytechnic University of Valencia hosted a webinar on this topic, that was then published on the Council of Coaches YouTube channel: https://www.youtube.com/watch?v=R3DYMhG59to (see Figure 17).

Figure 17: Screen capture of the FIWARE and universAAL webinar on YouTube (December, 2017). Link: https://www.youtube.com/watch?v=R3DYMhG59to

5.2 Workshop on United Agents: Interoperable Social Agents Frameworks Beyond SAIBA

This workshop will be held as part of the 20th ACM International Conference on Intelligent Virtual Agents (IVA) to be held online between October 19th and October 23rd, 2020.

Our partners, Sorbonne University (Catherine Pelachaud) and the University of Twente (Dirk Heylen, Dennis Reidsma) are organising a workshop at the 20th ACM International Conference on Intelligent Virtual Agents (IVA) Conference, titled: "Workshop on United Agents: Interoperable Social Agents Frameworks Beyond SAIBA".

The full list of workshop organisers is: Hendrik Buschmeier, Dirk Heylen, Catherine Pelachaud, Dennis Reidsma, Hannes Högni Vilhjálmsson.

Workshop's Summary
SAIBA, FML, and BML have been defined by the agent community over several workshops. They are used by different research groups. However, different research questions are arising, new technological developments have come up. There is the need to have architectures that allow for greater interactivity, for handling multiple agents, multiple users… There is a need to revise the common tools developed by the agent community. This workshop aims to address issues to go beyond SAIBA and move toward architectures that will allow modelling united agents on the same platform or not.

Learn more about the workshop here: https://iva2020.psy.gla.ac.uk/workshops/workshops/

5.3 Workshop: How to Evaluate Digital Health Innovation Still Under Development

This workshop will be held during the 16th World Congress on Public Health, which will be held online between October 12th and October 16th, 2020. The title of the workshop is: **How To Evaluate Digital Health Innovation Still Under Development. Three Examples Based On The Model Of Continues eHealth Evaluation.**

This workshop is organised by Roessingh Research and Development and chaired by Stephanie Jansen-Kosterink, PhD and will have a duration of 60 minutes.

The workshop will not only focus on Council of Coaches, but instead use the project as one of three cases to discuss how to evaluate digital health innovations that are in an early stage of development.

Objectives
Despite the common advantages of digital health, such as the increase of quality of care and the decrease in healthcare costs, the implementation of these innovations into public health are lacking. A positive impact in daily clinical practice is important for the successful implementation of digital health (Janssen & Johansen, 2018) (Broens, et al., 2007). Still it is difficult to assess the true impact of digital health in a real-life setting as proper evaluation studies are challenging (Lehoux, Vincent, & Visintin, 2009) (Laplante & Peng, 2011) (Ekeland, Bowes, & Flottorp, 2011). Within a clinical setting the focus of these evaluations are mainly on clinical effectiveness or quality of care. But these kinds of evaluations are only feasible when the technology is mature and no further development is needed. This is also indicated by the most popular method for the evaluation of digital health, based on the Health Technology Assessment (HTA) model. This method, the Model for Assessment of Telemedicine (MAST) (Kidholm, Clemensen, Caffery, & Smith, 2017) (Kidholm, et al., 2012) (Kidholm, Jensen, Kjølhede, Nielsen, & Horup, 2016), is only applicable for fully mature technologies. Unfortunately, MAST does not specify when a technology is mature enough. Resulting in evaluations of immature technologies in clinical trials. In our opinion, the maturity of the technology can be assessed based on the Technology Readiness Levels (TRLs). With TRLs we can clearly communicate whether a technology, such as a digital health intervention, is ready for use in daily clinical practice. An understanding of both the performance and the intended use of the technology is required to establish TRL and should be the starting point for every evaluation.

The objective of this workshop is to present our model for continuous eHealth evaluation (Jansen-Kosterink, Vollenbroek-Hutten, & Hermens, 2016) and to demonstrate it based on three evaluation studies of new developed digital health innovations for public health. All three innovations are still under development and have different TRLs. This will help other researchers to plan their evaluation studies with health technologies in an early phase of development. In our opinion the evaluation of digital health should be a continuous and iterative process, aligned to the progress in the development process itself. The presentations will each focus on the evaluation of a new digital innovation with real end-users and in a real-life setting. These evaluations are based on our model for continuous eHealth evaluation and will be presented as three separate cases.

We will conduct a 60-minutes interactive workshop, which starts with an overview presentation (15 minutes) of our model for continuous eHealth evaluation. Next, three presentations (3x 10 minutes) on

the evaluation of the three new digital health innovation will follow. The workshop ends with a plenary discussion (15 minutes) to discuss our view on the evaluation of digital health (model for continuous eHealth evaluation) and the development of the digital health innovations with the attendees.

Main messages

- The evaluation of digital health is challenging but the maturity of the technology must be the starting point of every digital health evaluation.
- The evaluation of digital health should be a continuous and iterative process, aligned to the progress in the development process itself.

Presentations: Next to the overview presentation (see workshop objective) the workshop consist of three presentations of each 10 minutes.

Presentation 1: The acceptance of clinical decision support systems among clinicians in the treatment of neck and/or back pain.

Background: Clinical Decision Support Systems (CDSSs) are computerized systems using case-based reasoning to assist clinicians in making clinical decisions. Despite the proven added value to public health, the implementation of CDSS clinical practice is scarce. Particularly, little is known about the acceptance of CDSS among clinicians. Within the Back-UP project (Project Number: H2020-SC1-2017-CNECT-2-777090) a CDSS is developed with prognostic models to improve the management of Neck and/or Low Back Pain (NLBP). Therefore, the aim of this study is to present the factors involved in the acceptance of CDSSs among clinicians.

Methods: To assess the acceptance of CDSSs among clinicians we conducted a mixed method analysis of questionnaires and focus groups. An online questionnaire with a low-fidelity prototype of a CDSS (TRL3) was sent to Dutch clinicians aimed to identify the factors influencing the acceptance of CDSSs (intention to use, perceived threat to professional autonomy, trusting believes and perceived usefulness). Next to this, two focus groups were conducted with clinicians addressing the general attitudes towards CDSSs, the factors determining the level of acceptance, and the conditions to facilitate use of CDSSs.

Results: A pilot-study of the online questionnaire is completed and the results of the large evaluation are expected spring 2020. Eight clinicians participated in two focus groups. After being introduced to various types of CDSSs, participants were positive about the value of CDSS in the care of NLBP. The clinicians agreed that the human touch in NLBP care must be preserved and that CDSSs must remain a supporting tool, and not a replacement of their role as professionals.

Conclusion: By identifying the factors hindering the acceptance of CDSSs we can draw implications for implementation of CDSSs in the treatment of NLBP.

Presentation 2: Evaluation of a digital health gamification platform with older adults: An observational cohort study with a pre-test/post-test design.

Background: Older adults are usually less physically active than younger adults. Physical inactivity can lead to frailty, which can increase the possibility of being admitted in a hospital and, being functional limited. To handle frailty Stranded is developed. Within this platform the user will be shipwrecked and has to build a boat to leave an uninhibited island. The user can leave the island by executing physical exercises and playing cognitive games. The primary aim focussed on differences in quality of life and perceived health status after using Stranded (TRL7). The secondary aim focussed on the usability of and user experience with Stranded. This study was conducted within the FRAIL-project (Eurostars-2 10.824).

Methods: An observational cohort study with a pre-test/post-test design was carried out. The pre-test measurements were performed before the use of Stranded, and the post-test measurements after using it for four weeks. The study population consisted of older adults, 55 years of age or older and each subject signed an informed consent form.

Results: One hundred and eleven older adults were included in this study (64.9% female and 35.1% male) and 91 participants started using Stranded. In total, 59 subjects dropped out. Two health variables significantly increased (n=52), the subjects' perceived health state on a visual analogue scale and the subjects' quality of life viewed from the positive health perspective. Stranded's usability scored an

average of 61.3 (SD=21.6). The average scores on the user experience domains were all between 3.3 and 3.9 on a 7-point scale. The subjects did not had a strong negative or positive opinion about these domains.

Conclusion: The average quality of life increased slightly. It is hard to find an appropriate population to investigate the effects of these innovations, because of not willing to include too frail older adults for whom participating could be intensive. The usability was perceived as acceptable.

Presentation 3: A first long-term evaluation of the Council of Coaches application: An observational cohort study.

Background: Due to the aging population, more and more older adults are living with chronic conditions. Adopting a healthy lifestyle can reduce the impact of these conditions. To address this issue, Council of Coaches (COUCH) has been developed within the Council of Coaches project (European Union's Horizon 2020 research and innovation program under grant agreement No. 769553). COUCH is a web application (TRL6), in which users can have virtual conversations with a group of embodied virtual coaches. These coaches give users information, feedback and tips on adopting a healthy lifestyle primarily in the areas of physical activity and nutrition. The aim of this evaluation is to assess the use, user experience, and potential health effects.

Methods: Use will be assessed by the log data of COUCH, and user experience and potential health effects will be assessed by questionnaires. The target population (older adults, 55 years or older) will participate in an observational cohort study with a pre-test/post-test design. Subjects will complete multiple questionnaires: before the baseline phase, after the implementation phase and, after the follow-up phase. Each participant will sign an informed consent form.

Results: The COUCH evaluation consists of two rounds. The first round started in February 2020, the second round will start in May 2020, with 25 subjects planned in each round. For the first round, 26 subjects are already recruited and included. For the second round, 12 subjects are already recruited. In August 2020, the results of the evaluation will be available. During this presentation, these results will be presented. This will consist of user demographics, use, user experience, and of the results on potential health effect.

Conclusion: This evaluation will give us a broad overview of the use, user experience and effectiveness of COUCH in a real-life setting, leading to further development of COUCH.

6 Council of Coaches in Educational Activities

The Council of Coaches project is innovative in the field of human-computer interaction and blends elements of health-, technology- and responsibility with its strong focus on principles of Responsible Research and Innovation (RRI). As such, the project offers great opportunities for training the next generation of innovators in these domains – the students of our universities and higher educational programs.

In this chapter we highlight how Council of Coaches has contributed to the training of students in various course programs (see Section 6.1) and individual student assignments (see Section 0).

6.1 Courses

In 2019, Mark Snaith from the University of Dundee (UDun) visited the University of Twente in the Netherlands to give a special guest lecture on "Dialogue and Argumentation" as part of the series of lectures given to Interaction Technology students following the course on Conversational Agents. The course materials that were created (see some slides in Figure 18 below) are ready to be re-used in next year's courses.

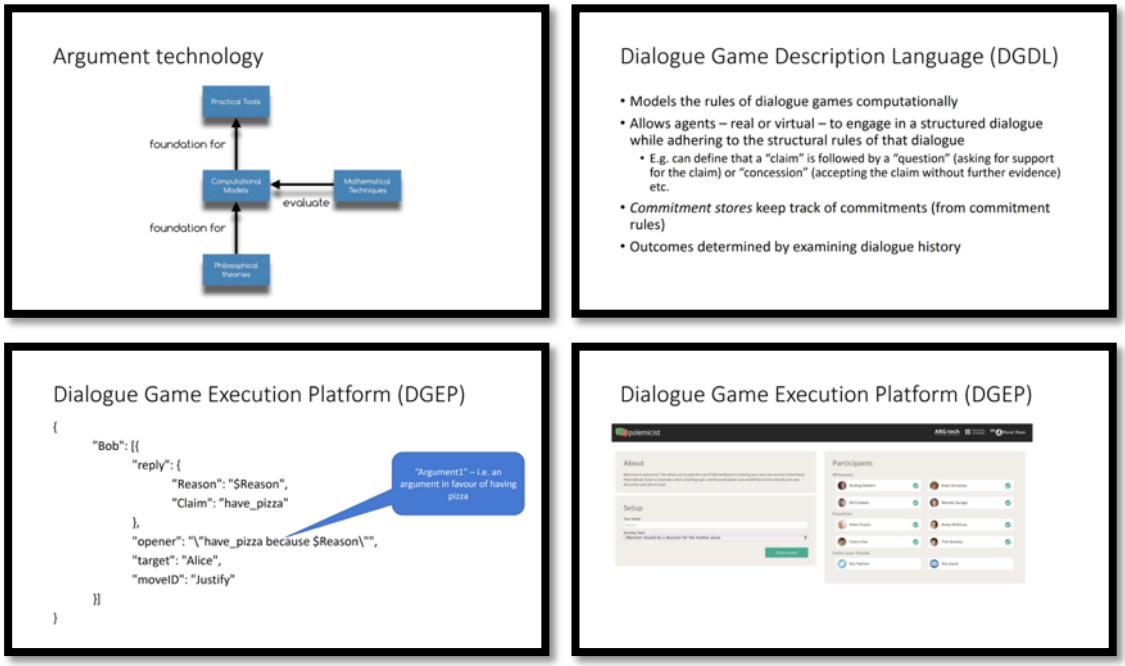


Figure 18: Slides from the Dialogue and Argumentation course taught by the University of Dundee as part of the Conversational Agents lecture series at the University of Twente.

At the University of Twente, in the course years of 2019 and 2020, the Agents United software framework was used as software platform for students to handle a practical assignment in the course "Foundations of Interaction Technology". The challenge for this assignment was to modify BML (Behaviour Mark-up Language) specifications to support multiple agents, using the Council of Coaches technical demonstrator framework. Specific training material was developed for this course in 2019 (and refined in 2020), in which the students also served as "documentation testers". Based on how successful students were able to setup and work with the framework given only the training material presented below, we were able to further improve this documentation (see Section 3 and Section 4).

Challenge Foundations of Interaction Technology

W7 - Behaviour Change Support System

INSTALL COUCH PROJECT (tested on Windows 10 64bits, should also work on macOS, but not tested in this setup) For questions please contact Randy Klaassen and/or Merijn Bruijnes.

Enough disk space (about 3GB)

Tools

- Unity3D (version known to work 2017.3.1f1, other 2017.x versions might work, versions known to NOT work 2018.x): https://unity3d.com/get-unity/download/archive
- Java: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Ant: https://ant.apache.org/
- ActiveMQ (for non windows users): http://activemq.apache.org/download.html

Installation

- Download the following .zip file: https://www.dropbox.com/s/6fukuu9cgebkadf/TechnicalDemonstrator-COUCH.zip?dl=0
- In a terminal (cmd in windows, terminal in apple), navigate to the folder \UT_HMI\HmiCouch
  - Type: ant resolve
  - ...wait patiently, this can take a while (about 2GB, depending on what is needed)...
  - Type: ant compile
- Open the unity3d project first time.
  - To open de project, open Unity, select 'Open', and navigate to and select the folder: \UT_HMI\UnityProject\CharacterCreatorNew
  - The first time this can take some time.
  - Open the scene.
    - In the project assets (standard location of this panel is at the left, bottom), navigate to \Assets\HMI\Scenes
    - Double click on the scene (standard location of the panel is at bottom): CouchProjectDemo.unity
- Now you should see the couch scene, some buttons and a table. Next time you open the unity project, it should automatically load the correct project scene.

Running

- Make sure ActiveMQ service is running. (in Windows you can Run ActiveMQ from \UT_HMI\Launchers\ActiveMQ_Start.bat (to check if it is running correctly, open a webbrowser and enter http://localhost:8161))
- Run ASAP, the behaviour planner, from \UT_HMI\Launchers\ASAP_Start.bat
  - Ready looks like: h.u.loader.UnityEmbodimentLoader - Waiting for AgentSpec…
- Open the unity3d project.
  - Press play and wait for the agents to take position behind the table.
- Run the ASAP BML window from \UT_HMI\Launchers\BmlWindow_ASAP_Start.bat
  - Run an example from the dropdown and press play

Stopping

- Unity: press the play button again (and close the unity editor).
- ASAP: select the cmd prompt window with ASAP and press ctrl+c

Behavior Markup Language (BML)

The official BML specification can be found here: http://www.mindmakers.org/projects/bml-1-0/wiki

In this challenge we will make use of a slightly modified BML specification to support multiple agents. Below we will present some examples and explain how this can be used in a multi-agent setup. The examples are also available in the BML ASAP window (see Figure XX)

Example: Multi-agent BML and synchronisation between the coaches

In the examples below we show you how we can define BML for the multi-agent setup. Each BML block should have the attribute characterId. CharacterId refers to one of the coaches available in the scene (in this case “COUCH_M_1”, “COUCH_M_2” or “COUCH_F_1”)

Verbal and non-verbal behavior of one agent can be synchronised within one agent or between the agents. In the example below COUCH_M_2 will start speaking after COUCH_M_1 finished his speech and waited 1 second (start="bml1:speech1:end+1" bml1 referes to the first BML block (bml1), speech1 refers to the first (and only) speech blok of bml1 and end+1 refers to the end of the speech of speech1 plus 1 second).

See the “speech” example (should be in the ASAP BML window dropdown menu).

```
<bml      id="bml1"      characterId="COUCH_M_1"      xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
 xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
      <speech id="speech1">
            <text>Hello there! I'm COUCH M 1</text>
</speech>
</bml>"


<bml      id="bml2"      characterId="COUCH_M_2"      xmlns="http://www.bml-
initiative.org/bml/bml-
1.0"        xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
<speech id="speech2" start="bml1:speech1:end+1">\n
<text>Hello there! I'm COUCH M 2</text>
</speech>
</bml>"


<bml      id="bml3"      characterId="COUCH_F_1"      xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
<speech id="speech3" start="bml2:speech2:end+1">
            <text>Hello there! I'm COUCH F 1</text>
</speech>
</bml>"
```

Example: Gaze and point

Gazing to objects in the world. The coaches can gaze to each other as follow:

```
<bml     id="bml30a"     characterId="COUCH_F_1"     xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
<speech start="1" id="speech1">
<text>An up, update!</text>
```

```
</speech>
</bml>


<bml id="bml30b" characterId="COUCH_F_1" xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
<gaze id="gaze1" influence="NECK" start="0" end="1"
target="head_COUCH_M_1" />
</bml>
```

In this example COUCH_F_1 will gaze at COUCH_M_1. She will start at time 0 and the gaze is ended at time 1. Targets to gaze at are head_COUCH_M_1, head_COUCH_M_2, head_COUCH_F_1, camera and BrowserGazeTarget.

Pointing at objects in the world can be done as the example below. How to control the content of the browser is shown in the mwe:sendJsonMessage block

```
<bml id="bml1" composition="REPLACE" xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
xmlns:mwe="http://hmi.ewi.utwente.nl/middlewareengine"
characterId="COUCH_M_1">
<speech id = "speech1">
<text>We also have a functional prototype that you can play with. As you
can see <sync id="s1"/>here, it looks very nice!</text>
</speech>
</bml>


<bml id="bml2" composition="REPLACE" xmlns="http://www.bml-
initiative.org/bml/bml-1.0"
xmlns:mwe="http://hmi.ewi.utwente.nl/middlewareengine"
characterId="COUCH_M_1">
<pointing id="point1" target="BrowserGazeTarget"
mode="LEFT_HAND" start="bml1:speech1:s1-1" stroke="bml1:speech1:s1"
end="bml1:speech1:s1+2"/>


<gaze id="gaze1" target="BrowserGazeTarget"
influence="NECK" start="bml1:speech1:start" end="bml1:speech1:s1+1"/>


<mwe:sendJsonMessage id="m1" start="0" end = "1"
middlewareloaderclass="nl.utwente.hmi.middleware.activemq.ActiveMQMiddlew
areLoader" middlewareloaderproperties="oTopic:COUCH/UI/URL">
{"url":"http://demo.council-of-coaches.eu"}
</mwe:sendJsonMessage>
</bml>
```

The characterId of the coaches. From left to right, COUCH_M_1, COUCH_M_2 and COUCH_F_1. The browser can be targeted as “BrowserGazeTarget”, the camera can be targeted with “camera”

FAQ answers:

- One speech behaviour in one BML block
- Other behaviours can be combined, per character

- Use unique BML block ids
- Use unique behaviour ids (within a block)

- Unexpected behaviour? Take a look at the feedback in the BML window
- Fixed your BML, but still no joy? Restart ASAP

6.2 Student Assignments

The Council of Coaches project has provided a great context for various students to perform thesis- or other assignments. Below we provide an overview of projects in which the Council of Coaches has served as training for these students.

6.2.1 Generation of Multi-Party Dialogues Among Embodied Agents to Promote Active Living and Healthy Diet for Subjects Suffering from Type 2 Diabetes Based on Theories of Behaviour Change and Behaviour Change Techniques

Student	Sneha Das
Study	University of Twente – Biomedical Engineering
Host Organisation	Roessingh Research and Development
Finalized Date	January 2019

Diabetes Mellitus is a chronic condition that is showing a rise in numbers in terms of prevalence and incidence across both the developed and developing countries of the globe. The condition also contributes towards other detrimental ailments such as cardiovascular diseases, blindness and gangrene in the foot region. This calls for health measures that would enable a proactive, self-management of the condition and aid the individual to lead a normal life and prevent further progression of Diabetes. With the advancements in the Internet and Internet of Things which has led to the development of e-health, the concept of personalized virtual coaching is being looked into. Since the concepts of being proactive and self-management are being discussed to cope with Diabetes, it is necessary for the individuals to feel inspired, motivated and actively involve themselves in the optimal decision-making process for the betterment of their health.

The work done during the course explores the mitigation factors that would control Diabetes Mellitus. It was found out that an active living and a good diet are the corner stones in the treatment of this condition. Thus, the Theories of Behaviour Change through the Behaviour Change Techniques are being applied to motivate the individuals to improve or change their lifestyle and(or) diet. The objective of the course is to develop dialogues based on the Theories of Behaviour Change and Behaviour Change Techniques, among embodied conversational agents who would play the role of various coaches focusing on virtual user-centric coaching to motivate the users to adopt a healthy lifestyle and diet.

Generation of Multi-Party Dialogues among Embodied Conversational Agents to Promote Active Living and Healthy Diet for Subjects Suffering from Type 2 Diabetes

Kuthethur Sneha Jagannath Das[1,2], Tessa Beinema[2], Harm op den Akker[2] and Hermie Hermens[1]
[1]*Biomedical Signals and Systems, University of Twente, 7522NB, Enschede, The Netherlands*
[2]*Telemedicine Group, Roessingh Research and Development,7522 AH, Enschede, The Netherlands*

Figure 19: Feature image for the assignment "Generation of Multi-Party Dialogues Among Embodied Agents to Promote Active Living and Healthy Diet for Subjects Suffering from Type 2 Diabetes Based on Theories of Behaviour Change and Behaviour Change Techniques".

Results / Outcomes

The Theories of Behaviour Change and the BCTs discussed under the section of Methods are the cornerstones on which the dialogue development has been implemented. The dialogues are written to promote the interaction among the ECAs and between the ECAs and the subject(user). The main objective of the dialogues is to motivate the subject to be physically active and adopt a healthy diet which in turn would result in preventing further progression of T2DM.

For the purpose of the generation of dialogues, three ECAs have been created who would be providing a virtual coaching to the subject. The virtual coaches have expertise in physical exercise and fitness training or are nutritionists. The third ECA is a volunteer who is also a diabetic subject but is dealing with the condition in a successful manner. In addition to the feature of virtual coaches, a family member(spouse) can also be a part of this coaching module. Since the spouse would be well aware of the subject's condition this option has been considered. Based on the Theories of Behaviour Change, BCTs and the ECAs, five dialogue sets were developed. The work performed in this student assignment resulted in the following publication:

> (Das, Beinema, op den Akker, & Hermens, 2019) Kuthethur Sneha Jagannath Das, Tessa Beinema, Harm op den Akker and Hermie Hermens, "Generation of Multi-Party Dialogues among Embodied Conversational Agents to Promote Active Living and Healthy Diet for Subjects Suffering from Type 2 Diabetes", in Proceedings of the 5th International Conference on Information and Communication Technologies for Ageing Well and e-Health – Volume 1: ICT4AWE, 297-304, 2019, Heraklion, Crete, Greece.

6.2.2 User-interface redesign for the coaching environment of Council of Coaches

Student	Casper Kroon
Study	University of Twente – Industrial Design
Host Organisation	Roessingh Research and Development
Finalized Date	November 2019

Roessingh Research and Development is working on the project Council of Coaches, a two-year running project to develop an application to assist people with chronic conditions in their everyday life. The idea is to create interactions with virtual coaches. One of the main goals for Council of Coaches is that the users can feel a personal connection with these characters. This report describes a design proposal for the interaction design and style of the Council of Coaches application, fitted to the specific target group of older adults using co-design methods. This design is one example of designing for a particular segment of the population of users, which eventually could be expanded to other user segments.

Figure 20: Feature image for the assignment "User-interface redesign for the coaching environment of Council of Coaches".

Results / Outcomes

In the first phase, the personalisation of the app and segmentation of the user groups that will be used in the design process were looked at. In the second part, earlier designs were evaluated using formative qualitative research methods and contextual inquiry. This was to see if the assumptions that were made concerning the design can be justified or improved for the target group of older adults. In phase 3, co-creation was performed in workshops with older adults in Enschede and Haarlem. It was found that the participants could not relate to the current style of Council of Coaches. They wanted to be taken seriously and did not receive this from the cartoon characters in the application and they found many attributes to bear little significance and to be distracting. This resulted in the realisation that a balance should be struck between formality and informality. In the fourth phase, the findings from the co-creation sessions were incorporated into two functional prototypes which differentiated in style (one contained a photorealistic style and one a drawn style) which were compared using a questionnaire.

In the final phase, a new design was created. This contained multiple variants of the user interface of the coaching environment and different interaction metaphors. Of these variants, multiple were incorporated into one functional prototype. In the discussion, it will be discussed which aspects of this design are generic and can translate to other user segments and which aspects are specifically for older adults. The approach in this report provides a thorough understanding of the intended target group, but is time-intensive, however, and possibly unrealistic if it has to be done every time for smaller user segments. In conclusion, this project presented an example of a personalized interface for a segment of the target group of Council of Coaches using principles of co-design. One of the most important insights was the importance of early testing of assumptions. To expand this approach to new segments, we need to take into account available resources. Therefore, a framework is recommended for designing new variations of the design for new user segments where the users can quickly be involved without taking much time.

6.2.3 Designing a Project Management Support Tool for development projects that are part of European eHealth research projects

Student	Vera van den Groenendal
Study	University of Twente – Industrial Design
Host Organisation	Roessingh Research and Development
Finalized Date	June 2020

The project management of a development or research project can be supported by project management tools. This master thesis focusses on "Designing a Project Management Tool that Supports the development project that is part of a European eHealth research project".

This thesis is executed at Roessingh Research and Development (RRD). RRD is involved in multiple European eHealth research projects where RRD is mostly responsible for the development of a service or product. The development of such service or product is based on the European research but can be seen as an individual development project. These European projects require collaboration with multiple partners, the internal stakeholders are often only partly involved in a project and all stakeholders have their own goals within the project. This complicates the project management. This thesis researches how to design a tool that can support the project management of the development part of a research project.

Council of Coaches is one of RRD's current European eHealth research projects. This project serves as case study for the development of the Project Management Support Tool (PMST). The PMST is proposed to RRD to support their European research projects in the eHealth department.

The project management of RRD's research projects is analysed to determine strengths, weaknesses and opportunities. The PMST's potential lies in supporting the project management team, improving project documentation, stimulating stakeholder collaboration, supporting project communication, supporting the decision-making process, handling emerging new requirements and supporting effective and efficient knowledge sharing.

The Project Management Support Tool is IT-based and supports multiple projects with their project management at the same time. It is possible for the users to use the PMST for multiple projects simultaneously. All the user input is saved and the PMST builds up a database. The available data is combined into understandable and relevant information for the user.

The PMST includes characteristics of Synthetic Environments (SE) to structure these complex research projects and to support the stakeholders with their project activities. A SE is a combination of virtual and real elements describing an alternative setting of a real-life situation. In this case the PMST creates an environment for the project that has SE characteristics. The PMST visualises the influence of a project update in the project overview, which allows the user to evaluate this change and decide whether or not to implement the update. Building a SE focusses on involving all stakeholders, including their opinions and documenting their requirements and decisions.

In the PMST the internal stakeholders of a project can document decisions and requirements. The users (who are internal stakeholders of a project) add information to the PMST, and the PMST presents this information in clear overviews and lists. Included are the rationale behind a requirement/decision, which requirement/decision is related to what project aspects/requirements/decisions, which stakeholders are involved in a certain project task and the author. The project overview is based on actor networks and includes all stakeholders, project aspects, requirements and decisions mentioned. Filters and graphical elements support the visualisation and can present information in different structures. This allows the user to analyse information from different perspectives. The PMST personally supports each stakeholder with notifications about requested actions and actions of other users.

Integrating and combining characteristics of Synthetic Environments, requirement engineering and decision-making support systems into the designed PMST has proved to be of added value. All decisions and requirements can effectively and efficiently be documented and managed in the PMST. This allows for easy adding and updating of certain decisions or requirements. By adding more information to the PMST, the PMST reinforces over time.

A feature to set goals for and evaluate the project (management) is included, this will provide very useful insights to the internal stakeholders that they can use in their (next) projects. Next to that, the PMST present an overview of all RRD's research projects, visualising how these projects are related and which stakeholders are involved.

A mock-up of the PMST is designed to provide an example of how the PMST could look like and function. The PMST design is positively evaluated by the potential users. Thus, when the PMST will be developed according to the proposed design, the project management of RRD's research projects will be improved and the stakeholders will be better supported. With the PMST the project management of a development project within European eHealth research project will be supported and reach maturity and grow excellence over time.

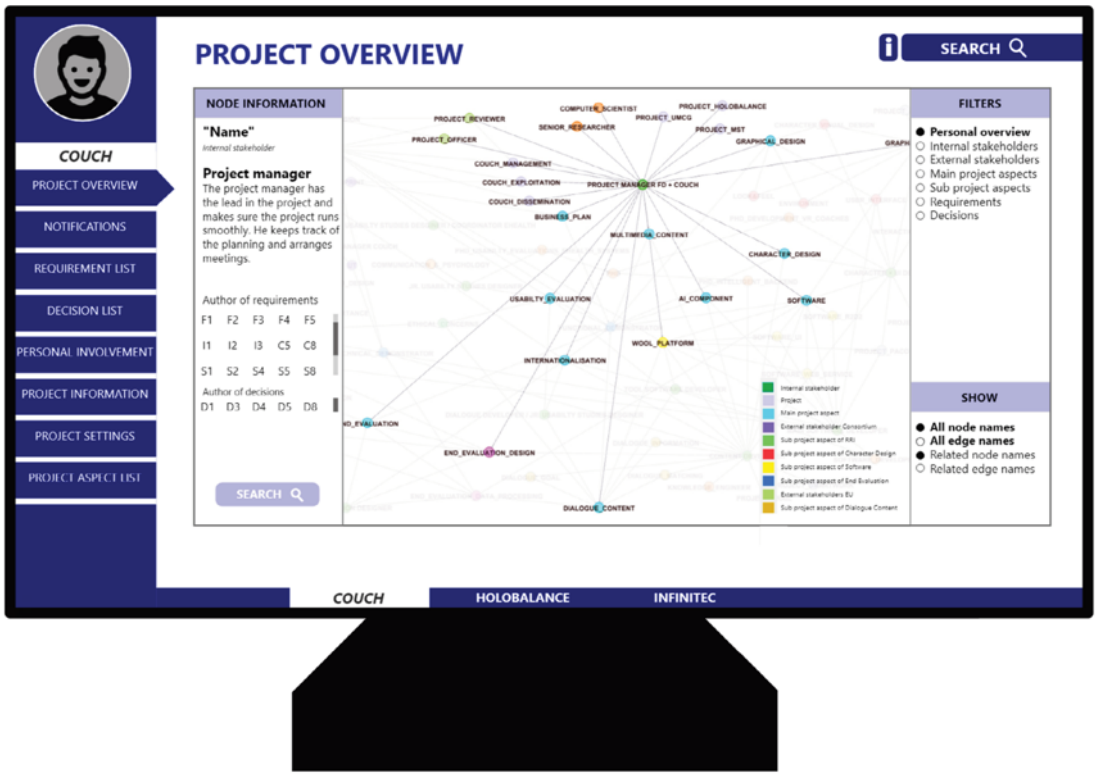


Figure 21: Feature image for the assignment "Designing a Project Management Support Tool for development projects that are part of European eHealth research projects".

Results / Outcomes

Integrating characteristics of Synthetic Environments (SEs) into the designed Project Management Support Tool (PMST) has proved to be possible. In SEs, blueprints are the backbone, they support stakeholders with categorising their information. This is integrated in the PMST by introducing project categories and main project aspects. The categories help classifying the information that added into the PMST by the user. The main project aspects are determined by the project management team and support the user with defining (sub) project aspects.

All stakeholders, project aspects, requirements, decisions plus their explaining information are brought together in an architecture. This architecture is based on actor networks and includes all elements (stakeholders, project aspects, requirements and decisions) mentioned. Filters and graphical elements support the visualisation and present information in a different structure. This allows the user to analyse

information from different viewpoints (perspectives). The disagreements between stakeholders or stakeholder groups should be discussed and the agreements represent a Synthetic Environment of Council of Coaches that can be developed in the PMST.

All decisions and requirements can be effectively and efficiently documented and managed in the PMST. This allows for easy adding and updating of certain decisions or requirements. A template is presented that asks the user to fill in certain aspects about a requirement and decisions. This information is documented and contains the knowledge that is saved by the PMST. Adding more information to the PMST reinforces the PMST over time.

All project elements can now be visualised in the PMST in a clear and understandable way. Adaptations can be made within only few steps, which allows the user to analyse the impact/influence of a project update.

To conclude, the PMST facilitates an approach to document project information (e.g. goals, approaches, stakeholders, requirements, decisions and rationales). It supports the decision-making process and gently forces the user to think of the impact of a certain decision. It handles emerging new requirements and requirement updates to allow requirement evolvement over time. By providing all information at one place (in the PMST) and sending notifications about actions, the PMST supports effective and efficient knowledge sharing.

The potential users who evaluated the PMST were positive about its functions and features, and stated improvements that are related to the (sub) goal(s). As a result, it is expected that the PMST would reach its goal and sub goals when designed and implemented.

It is expected that the Project Management Support Tool is broadly applicable and could provide support to development projects in general.

6.2.4 Internship(s) on Head Movement Analysis

Student	Céline Caristan & Amandine Guillin
Study	University of Paris
Host Organisation	Sorbonne University
Finalized Date	?

Two BSc students from the University of Paris performed internship assignments in the context of Council of Coaches. The objectives of the internships (BSc) were to:

- Annotate the video corpus recorded in the context of WP5 by the University of Dundee in order to transcribe head movements of participants.
- Conduct statistical analysis related to group behaviour and cohesion.
- Develop a 3D Unity environment

Results / Outcomes
Correlation was found between high cohesion and number of head movements and smiles.

6.2.5 Modeling Interpersonal Interactions in Multi-Party

Student	Julien Defiolles
Study	Sorbonne University
Host Organisation	Sorbonne University
Finalized Date	?

The objective of the internship is to develop an interaction model for a group of social agents. This objective is divided into two distinct parts:

1. Analyse models of interpersonal interaction emerging in multi-party interaction of humans.
2. Propose new measures to calculate interpersonal synchronization in a group of human agents.

During the 6-month internship, steps 1 and 2 were completed. First of all, by discovering the various works already carried out on interpersonal interactions in a group of agents, both in computer-related work concerning more than specifically the analysis and evaluation of these interactions only in the case of psychological research into the explanation of these behaviours. Then, with this understanding how a group works, how it is organized and the analysis of the different existing methods for measuring interpersonal synchrony, one can propose a brand-new measure of group synchrony from different voice parameters.

Figure 22: Feature image for the assignment "Modeling Interpersonal Interactions in Multi-Party".

Results / Outcomes

Synchrony in multi-party can be computed through four parameters: latency rate, pause duration, speaking time and silence, each adding and improving the accuracy of the calculation.

In order to obtain a calculation of group interpersonal synchrony, it was necessary to first learn the different definitions of the terms of the subject and know the different work already done on it. Thus, the reading of various articles and books on the group psychology, on the different computer or psychological methods for calculating the synchrony, was the beginning of the work.

To work on interpersonal synchrony and get results for to test it afterwards, a video corpus was needed. The choice was to take one already belonging to the team: the 'council of coaches' corpus. Then, we had to search for the different parameters to calculate the synchrony, we took some audio parameters to start with, the choice was made on the pause time, speaking time, silence, latency and backchannels. A backchannel is a short utterance produced by one person in the conversation while another person in the conversation is talking, in order to show interest. In order to calculate these different parameters, it was necessary to use a tool to extract prosody settings from the corpus videos, allowing to calculate the data mentioned above. For this we used the open source software OpenSmile.

Based on the different parameters calculated, it was necessary to mix the various parameters to obtain a value for interpersonal synchrony. Interpersonal synchrony is an important value in determining whether a group can communicate fluidly and dynamically. The synchrony described here is thus divided into four parts: latency rate, break time, speaking time and silence, each adding and improving the accuracy of the calculation. The set of tests shows that the synchrony obtained is independent of the number of people and videos used, and that the calculation is in sync.

6.2.6 The effect of an intervention with virtual coaches on the physical activity of adults aging 55 years or older

Student	Katrien Fischer
Study	Vrije Universiteit Amsterdam, Faculty of Behavioural and Movement Sciences
Host Organisation	Roessingh Research and Development
Finalized Date	June 2020

With an increasing life expectancy of the older adult population, being physically active is very important. Maintaining a healthy lifestyle can be supported by virtual coaches. It can change the motivation of a user, their behavior, beliefs or attitudes. The Council of Coaches' functional demonstrator, a web-based eHealth application, will be evaluated within this study, with a focus on the virtual physical activity coach, Olivia Simons. The aim of this assignment is to assess the effect of a virtual physical activity coach on physical activity levels by implementing the functional demonstrator in a real-life setting among older adults aging 55 years or older.

Results / Outcomes

This assignment was part of the final evaluation of the functional demonstrator, and focuses on Fitbit step data of the baseline week (1 week) and implementation phase (4 weeks) of the first round, and on Olivia (the physical activity coach). The study population consisted of 25 Dutch older adults, with a mean age of 66.5 years old (SD=7.5). With the collected Fitbit data, the average number of steps per day in all five weeks was calculated, see Boxplot below. The median steps per day in each week fluctuated around 9000 steps. With the Friedman test, no significant result was found between the different weeks.

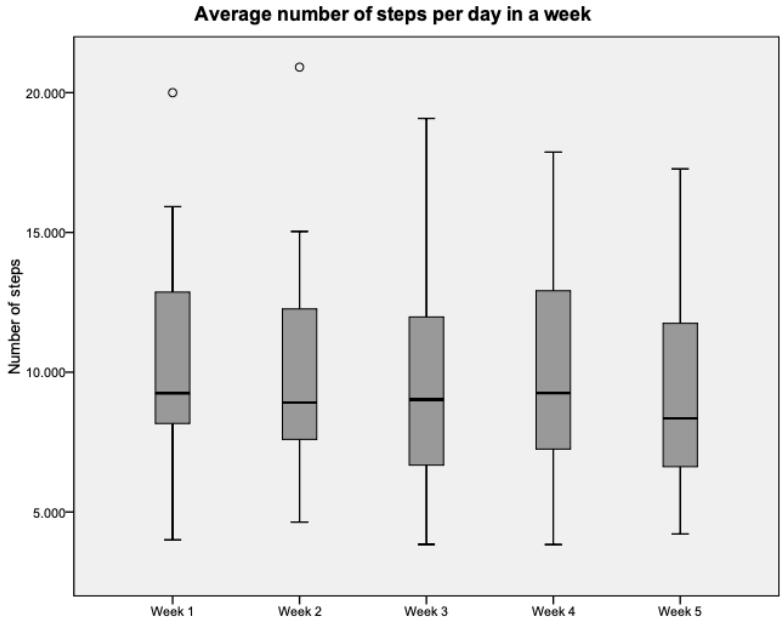


Figure 23: Feature image for the assignment "The effect of an intervention with virtual coaches on the physical activity of adults aging 55 years or older".

One participant did not interact with Olivia during the implementation phase. During the first week of the implementation phase, most participants interacted with Olivia, and the mean duration of the interaction was highest, namely 14.7 minutes (SD=18.1). The mean duration of the interaction with Olivia was lowest during the third week of the implementation phase: 4.7 minutes (SD=4.6).

The following factors for interacting with Olivia were mentioned:

- Health pressure (N=14): wanted to receive advices, ideas on how to stay physically active.
- Enjoyment (N=9): interested in what Olivia had to offer them.
- Positive health (N=4): wanted to be conscious of how to get a healthy body.

In conclusion, this study did not show that physical activity of adults aging 55 years or older was stimulated by an intervention with virtual coaches. The number of steps did not significantly increase and older adults will use a virtual physical activity coach mainly for health pressure. The virtual coach makes users more aware of daily movement and confronts and stimulates them. However, future studies are needed, in which it is important to make sure that the participants cannot receive notifications or feedback on the Fitbit. Also, a follow-up study must take participants' fitness into account and find out the findings for the long-term of using an e-health application.

6.2.7 Adding Speech to Dialogues with a Council of Coaches

Student	Laura Bosdriesz
Study	University of Twente, Interaction Technology
Host Organisation	Roessingh Research and Development
Finalized Date	*In progress.*

Currently in COUCH the user interacts with the coaches by selecting one of several predefined multiple-choice responses on a tablet or computer interface. Although this is a reliable way to capture input from the user, it may not be ideal for COUCH its target user group of older adults. Spoken dialogues might improve the user experience of COUCH for older adults.

In this project the COUCH system is adapted to support spoken interactions. Based on the research topics, the choice is made to use the NLSpraak speech recognizer for understanding the older adult's speech. In addition to incorporating automatic speech recognition, smart ways are investigated to organise the dialogue to facilitate adequate recognition in noisy and uncertain settings while keeping the conversation going. Finally, the user experience and the quality of dialogue progress in various settings is evaluated and compared to the original COUCH system and thereby a conclusion will be drawn about the robustness of state-of-the-art speech recognition in home settings.

Results / Outcomes
The assignment is still going on, so no final conclusions can be drawn yet. So far, the work has focused on the implementation of the NLSpraak speech recognizer and the Google text-to-speech synthesizer. First there has been looked into ways to deal with the spoken speech, starting with the recognizer listening to the first word of a sentence. This has been changed to the last word of a sentence and in some cases, manually adding keywords to the WOOL dialogue files. Furthermore, strategies to organise the interaction in cases when the speech is not appropriately recognized are implemented.

6.2.8 Developing a Virtual Social Coach

Student	Ellis oude Kempers
Study	University of Twente, Public Administration
Host Organisation	Roessingh Research and Development
Finalized Date	*In progress.*

One of the coaches of the Council of Coaches (COUCH) is a social coach, the goal of this coach is to make the user think about and work on his social network in order to stimulate a healthy lifestyle. Within the final demonstration of COUCH, 25 users get in touch with the virtual social coach for four weeks. After this demonstration, interviews with the users, questionnaires and the data of the desktop version will provide an oversight in how these users will be coached virtually to feel motivated to maintain their social network.

This research within COUCH together with the literature review will make it able to answer the following question: *How could a virtual coach motivate elderly to maintain a vital social network?* To answer the question, the research will focus on two aspects: the way in which the technology should be shaped and the required coaching characteristics of the social coach.

6.2.9 The Embodiment Game: A Methodology to Study the Effect of Embodiment in Human-Machine Interaction

Student	Sara Falcone
Study	University of Trento, Centro Interdipartimentale Mente/Cervello - CIMEC
Host Organisation	University of Twente
Finalized Date	*July 2019*

In the context of human-agent interaction, embodiment plays an important role in how the interaction is perceived. This work explored the use of "Echoborgs" as a method for investigating how social perception is related to the interface or form of the embodiment. An Echoborg is a conversational partner (human or agent) who is secretly shadowing the responses of a different third party (human or agent). In other words, a human actor having a conversation with a subject can be "controlled" by a virtual agent, behaving in the same way that the artificial intelligence would, or conversely, a virtual agent can be controlled by a human actor, behaving in the same way a human would.

Results / Outcomes
This thesis documents the explorative work on operationalizing and implementing this approach using the Council of Coaches platform. The topic of conversation was an ethical debate on the trolley dilemma with three assigned roles: moderator of the debate, proponent (for pulling the lever) and opponent.

A multimodal embodied conversational agent was created with behavior and dialogue informed by the dialogue and motion in an analogous human-human interaction. Vice versa, an interface (that goes beyond a radio earpiece) was created to train a human to act as an Echoborg to carry out the behaviour generator and dialogue system of a virtual agent.

Participants partook in the ethical debate either with two agents, or with one agent and one echoborg. A corpus on interaction scenarios was collected that allows some comparisons between the same dialogue system on agents and Echoborgs.

Figure 24: Feature image for the assignment "The Embodiment Game: A Methodology to Study the Effect of Embodiment in Human-Machine Interaction".

6.2.10 Implementing Virtual Reality in the Council of Coaches system

Student	Rens van der Werff
Study	University of Twente, Interaction Technology
Host Organisation	-
Finalized Date	*July 2020*

As the population ages the demand for personalised healthcare coaches grows larger. To keep up with this demand, the Council of Coaches (COUCH) was created, aiming to provide personalised coaches from at home, by means of an interactive virtual conversation with some coaches. In order to keep the users using the system and its advice credible, user engagement is important. One way of possibly increasing user engagement is by implementing Virtual Reality (VR) into the system. To find out what

features work well in VR in COUCH, two prototypes were created that differentiated in seven different areas, from location to interaction. These two prototypes were each tested by using 6 participants that served as proxy users and 2 target group users that were approached online with an interactive video in order to adhere to the COVID-19 guidelines.

Figure 25: Feature image for the assignment "Implementing Virtual Reality in the Council of Coaches system".

Results / Outcomes
The results of these tests showed that the two features that have the most impact on user engagement are the environment; a cosy room worked best in this research, and accessibility; here shaped as subtitles to support the spoken text. A small sample size means that more research on the topic is recommended and more research with the target group should be performed.

6.2.11 Council of Coaches in Virtual Reality

Student	Timo Petersen
Study	University of Twente, Interaction Technology
Host Organisation	-
Finalized Date	*July 2020*

This research aims to determine how a new 3D modelled environment can influence the engagement of older adults with the Council of Coaches system in Virtual Reality (VR). Four prototype environments were made based on insights from related work and literature, early user confrontation and user scenarios. The prototypes include an indoor beach house environment and an outdoor forest environment, which can be experienced in traditional screen-based (2D) interactions and through a VR headset (3D). Three studies were conducted to evaluate the effect of the different components in the screen-based prototypes, the difference in experience between the screen-based and VR prototypes and the effect of the different components in the VR prototypes.

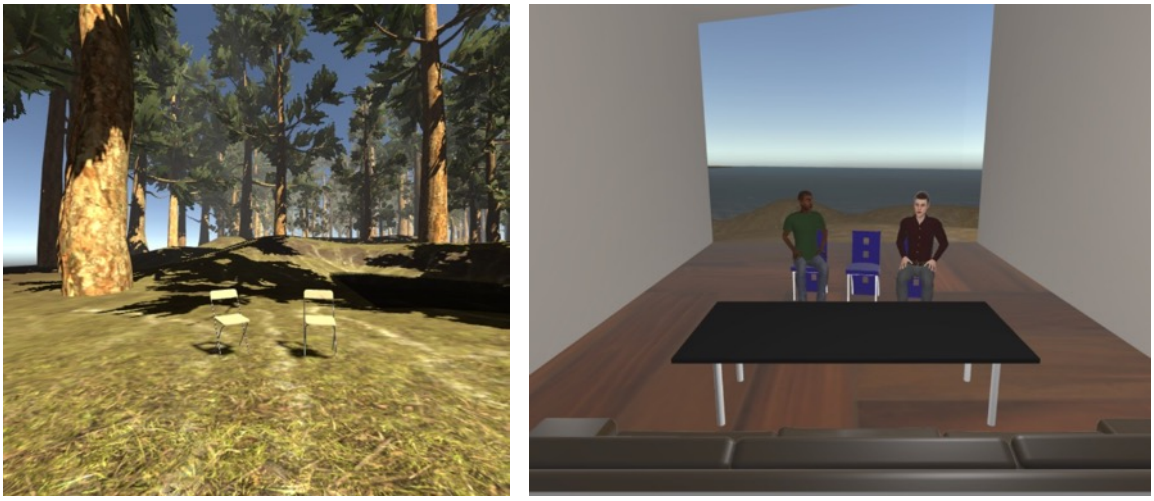


Figure 26: Feature image for the “Council of Coaches in Virtual Reality” assignment.

Results / Outcomes

Analysis of the three studies demonstrated a recommendation for the forest environment in VR. The forest environment was preferred because of the looks, feeling and view. The VR was preferred because it was seen as more realistic, giving a better experience and more immersion. The participants preferred the outdoor experience because of the unlimited view. This research found that the engagement of older adults with the COUCH system in VR can be influenced by the use of different environments, sounds, interaction mechanisms, screen-based or VR experience, indoor or outdoor experience, the use of a couch, table and different chairs.

7 Bibliography

Janssen, G. C., & Johansen, M. (2018). Factors Determining the Success and Failure of eHealth Interventions: Systematic Review of the Literature. *Journal of Medical Internet Research, 20*(5), e10235.

Broens, T., Huis in't Veld, R., Vollenbroek-Hutten, M., Hermens, H., van Halteren, A., & Nieuwenhuis, L. (2007). Determinants of successful telemedicine implementations: a literature study. *Journal of Telemedicine and Telecare, 13*(6), 303-309.

Lehoux, K., Vincent, C., & Visintin, M. (2009). A systematic review of clinical outcomes, clinical process, healthcare utilization and costs associated with telerehabilitation. *Disability and rehabilitation, 31*(6), 427-447.

Laplante, C., & Peng, W. (2011). A systematic review of e-health interventions for physical activity: an analysis of study design, intervention characteristics and outcomes. *Telemedicine Journal and e-Health: the official Journal of the American Telemedicine Association, 17*(7), 509-523.

Ekeland, A., Bowes, A., & Flottorp, S. (2011). Methodologies for assessing telemedicine: a systematic review of reviews. *International Journal of Medical Informatics, 81*(1), 1-11.

Kidholm, K., Clemensen, J., Caffery, L., & Smith, A. (2017). The Model of Assessment of Telemedicine (MAST): A scoping review of empirical studies. *Journal of telemedicine and telecare, 23*(9), 803-813.

Kidholm, K., Ekeland, A., Jensen, L., Rasmussen, J., Pedersen, C., Bowes, A., . . . Bech, M. (2012). A model for assessment of telemedicine applications: mast. *International Journal of Technology Assessment in Health Care, 28*(1), 44-51.

Kidholm, K., Jensen, L., Kjølhede, T., Nielsen, E., & Horup, M. (2016). Validity of the Model for Assessment of Telemedicine: A Delphi Study. *Journal of Telemedicine and Telecare, 24*(2), 118-125.

Jansen-Kosterink, S., Vollenbroek-Hutten, M., & Hermens, H. (2016). A renewed framework for the evaluation of telemedicine. *Proceedings of the 8th International Conference on eHealth, Telemedicine, and Social Medicine: eTELEMED.* Venice, Italy.

Das, K., Beinema, T., op den Akker, H., & Hermens, H. (2019). Generation of Multi-Party Dialogues among Embodied Conversational Agents to Promote Active Living and Healthy Diet for Subjects Suffering from Type 2 Diabetes. *Proceedings of the 5th International Conference on Information and Communication Technologies for Ageing Well and e-Health - Volume 1: ICT4AWE*, (pp. 297-304). Heraklion, Crete, Greece.

WOOLPlatform.EU. (2020, August). *WOOL GitHub Pages*. Retrieved from https://github.com/woolplatform/wool

WOOLPlatform.EU. (2020, August). *WOOL Online Editor*. Retrieved from https://www.woolplatform.eu/editor/wooleditor/app/

Secret Lab. (2020, August). *Yarn Spinner*. Retrieved from Yarn Spinner: https://yarnspinner.dev/

Twine. (2020, August). *Twine / An open-source tool for telling interative, nonlinear stories*. Retrieved from Twine / An open-source tool for telling interative, nonlinear stories: https://twinery.org/

8 Annex A: WOOL Language Documentation

8.1 Basics & Terms

A WOOL dialogue definition is essentially a definition of a series of dialogue steps (that we refer to as nodes) linked together through user replies.

We define the following terms:

- **Node** — A dialogue step that contains one Statement and a one or more Replies.
- **Statement** — Something an agent says.
- **Reply** — A possible reply that a user of the system can give.
- **Agent** — A virtual speaker within a dialogue.

8.2 WOOL Nodes

A **Node** consists of two parts, a `header`, and a `body`.

8.2.1 Header

The header consists of a series of lines, each with a `{key: value}`-pair. The two required key-value pairs are:

- `title` — a String that uniquely identifies this **Node** within this WOOL dialogue.
- `speaker` — a String that defines the name of the **Agent** speaking in this **Node**.

Note: this is a major difference from Yarn. In WOOL, each node represents a single step in a dialogue, and thus belongs to a single speaker (whereas Yarn allows multiple speakers and statements in the same Node).

You are free to define other key-value pairs that might serve as meta-data in your application. The WOOL Editor uses the following additional ones:

- `colorID` — a number between 0 and 9 specifying a colour. This is also used by the editor's run feature to allow multiple backgrounds, one for each colorID.
- `position` — used to position the nodes in the editor.

Below is an example of what this looks like in a .wool file:

```
title: Start
speaker: Robin
position: -416,112
colorID: 3
```

8.2.2 Body

The body of a **Node** contains at least one **Statement** and zero or more **Reply**s. A very basic example is given below:

```
Hello, my name is Robin!

[[Nice to meet you Robin!|NodeRobin2]]
[[Goodbye.|NodeEnd]]
```

This **Node** defines a **Statement** "*Hello, my name is Robin!*", uttered by the **Agent** "*Robin*" and two possible **Reply** options. When a user selects "*Nice to meet you Robin!*" he will be forwarded to a **Node** labeled "*NodeRobin2*", and when he selects "*Goodbye.*" he will be forwarded to the **Node** labeled "*NodeEnd*".

8.2.3 File Format

A `.wool` file consists of a list of concatenated Nodes separated by `===` markers. The header and body of the **Node** are separated by the `---` line. For example:

```
title: Start
speaker: Robin
position: -416,112
color: cyan
---
Hello, my name is Robin!

[[Nice to meet you Robin!|NodeRobin2]]
[[Goodbye.|NodeEnd]]
===
title: NodeRobin2
speaker: Robin
position: -216,112
color: red
---
Nice to meet you too, how are you doing?

[[I am fine and you?|NodeRobin3]]
[[Goodbye.|NodeEnd]]
===
...
```

8.2.4 Comments

If you want to document your WOOL scripts with comments, WOOL Supports line-comments. Everything after a double-slash `//` is considered a comment:

```
Robin: Thank you very much. // Thank the user for the gift.

// If it was a nice gift...
<<if $giftNice >>
   ...
```

8.3 WOOL Dialogues

A series of WOOL **Node**s is called a WOOL Dialogue. The following rules apply to WOOL Dialogues:

- All **Node** title's must be unique within a WOOL Dialogue.
- There must be one **Node** with the title "*Start*" (this is the default starting point of the dialogue).
- WOOL Dialogue files may contain letters, numbers, dashes and underscores, and end with `.wool`, valid examples include:
  - mydialogue.wool
  - my-dialogue.wool
  - my_dialogue-1.wool
  - 123dialogue_for-Robin.wool

8.3.1 Starting a Dialogue

Every WOOL Dialogue script must include a Node with title "Start". Applications that execute WOOL scripts can choose this as the default starting node for a conversation (or ignore it, and start somewhere else). For example:

```
title: Start
speaker: Robin
position: -416,112
color: cyan
---
Hello, my name is Robin!

[[Nice to meet you Robin!|NodeRobin2]]
[[Goodbye.|NodeEnd]]
===
```

8.3.2 Ending a Dialogue

There are two ways a WOOL dialogue can end:

- The user doesn't have any **Reply** options in the **Node** (the **Agent** has the last say).
- The user chooses a **Reply** option that leads to a **Node** with title "End" (user has the last say).

Example 1:

```
It was nice talking to you, bye!
```

Example 2:

```
Do you have any other questions?

[[I have nothing left to say.|End]]
```

Unlike the "Start" **Node**, the "End" **Node** is not mandatory to include in your WOOL Dialogue, as there are other ways to end the conversation. There is also nothing stopping you from creating a dialogue that can only loop indefinitely. When creating WOOL-based applications, you can also choose to provide a User Interface element that can "cancel" a dialogue at any time.

Note: the node with the Title "End" is thus treated as a special case. When other nodes refer to it, this Node should be created in the dialogue as usual, however its contents must be empty. When the "End" node is reached in a WOOL Dialogue, the application should simply "close" the conversation. The special meaning of Nodes titled "Start" and "End" is different from Yarn, and are marked as such in the WOOL Editor.

8.4 WOOL Statements

Ultimately, every WOOL **Node** should output some text to display to the user, but WOOL **Statement**s allow for a lot of flexibility in structuring and personalizing your dialogue.

8.4.1 Basic Statements

The most basic **Statement** is a simple line of text, that is uttered by the speaker (**Agent**) of the **Node**:

```
Hello, how are you?
```

8.4.2 Basic Statements with Variables

You can use variables within your **Statement**s. Variables start with a `$`-sign, followed by one of `A-Z a-z` and then any number of `a-z A-Z 0-9` or `_` (underscore). In short: start with a letter, then use letters, numbers or underscores, for example:

- $variableName
- $variable_name
- $var123

These variables can be used within **Statement**s to inject their values into a sentence, like so:

```
Hello $userFirstName, how are you?
```

8.4.3 Basic Statements with Special Characters

What if you actually want to include a $ character in your text? If it's not followed by A-Z a-z, you can just type $. But otherwise you can escape it with a backslash: \$. And to include a backslash? Just escape it with another backslash: \\. In fact you can escape any character with \ and it will not be treated as a special character. Some more examples: \<< \>> \[[\]]

8.4.4 Basic Statements with Mark-up

For the rest, WOOL doesn't care about any mark-up you might want to apply, if you want to add HTML tags around text, please go ahead. The parsers will ignore it, and simply output the text including mark-up to your application:

```
Hello <b>$userFirstName</b>, how are you?
```

8.4.5 Control Statements: Setting Variables

WOOL allows you to set variables using the `<<set>>` **Statement**:

```
<<set $userFirstName = "Bob">>
<<set $points = 0>>
<<set $hasReplied = true>>
```

The example above shows the three most common cases for setting either `String`, `number` or `boolean` variables. However, WOOL is much more flexible, and allows for example the set-**Statement**s below:

```
<<set $points = $points + 1>>
<<set $name = $firstname + " " + $lastname>>
<<set $string = "String" + 12345>> // $string is set to "String12345"
<<set $string = 1 + 2 + "3">> // is parsed from left to right, $string =
"33"
<<set $string = 1 + (2 + "3")>> // using brackets, resulting in $string =
"123"
```

As you can see, you don't have to define the `type` of the variable manually. Be careful when using more complex statements though. For example, when trying to add up numbers with Strings, WOOL will treat the result as a String.

8.4.6 Control Statements: Conditionals

WOOL supports if-then-else **Statement**s. The simple example:

```
<<if $dayPart == "Morning" >>
   Good morning ladies and gentlemen!
<<elseif $dayPart == "Afternoon" >>
   Good afternoon peoples!
<<else>>
   Good evening everyone!
<<endif>>
```

Please note that "==" is treated as strictly-equals.

WOOL also supports nesting these if-statements, if needed:

```
<<if $dayPart == "Morning" >>
    <<if $userFriendly == true>>
        Good morning, sir! How are you today.
    <<else>>
        Mornin'.
    <<endif>>
<<elseif $dayPart == "Afternoon" >>
   ...
```

Note that in the case of boolean variables (`$userFriendly`), you can leave out the " `== true`" part. E.g. the following is valid and will work as expected if `$userFriendly` is an actual `boolean` value:

```
<<if $userFriendly>>
```

However, if `$userFriendly` is actually a String with value "*No, he is not friendly.*", this expression will evaluate to `true`. If the variable is empty (or "unset"), the expression will evaluate to `false`.

Be careful with using the short-hand form `<<if $variableName>>`, because it is *not strict*, while for example `<<if $variableName == false>>` is strict. This means that if the variable `$variableName` has not been assigned a value, the following will happen:

- `<<if $variableName>>` will evaluate to `false`
- `<<if $variableName == false>>` will also evaluate to `false` (which may be counter intuitive)

If in your application you cannot be sure whether or not boolean variables have been assigned a value, our advice is to always use `<<if $variableName == true>>`, to avoid confusion.

8.4.7 Control Statements: User Interface Actions

Sometimes you might want to couple some event or action to a statement uttered by a speaker. WOOL supports specifically images, video or *other* generic actions.

The image example:

```
And here you can see a picture of a dog.
<<action type="image" value="dog.png">>
```

The video example:

```
I would like to show you this cool video I found.
<<action type=”video” value=”https://www.youtube.com/watch?v=dQw4w9WgXcQ”>>
```

The "generic action" example:

```
Let me show you something in this book I found.
<<action type="generic" value="OPEN_RECIPE_BOOK">>
```

An example with specified delay:

```
Okay, I'll take you to the puzzle game, just a second...
<<action type="generic" value="LAUNCH_PUZZLE_GAME" delay="10">>
```

In this example, the `delay` parameter defines that the action should be executed with some delay (whether you interpret this as 10 seconds, 10 milliseconds or 10 days is up to your UI developer).

Finally, you can add other parameters to action-**Statement**s, so as long as you don't call them type, value or delay you should be fine:

```
Okay, I'll take you to the puzzle game, just a second...
<<action type="generic" value="LAUNCH_PUZZLE_GAME" delay="10"
difficulty="medium" background-color="#0000FF" num_players="2">>
```

8.5 WOOL Replies

Every **Node** can define zero or more (indefinite, but please consult your UI designer) **Reply** options, the different types are defined below.

8.5.1 Basic Replies

The standard Reply option defines a "user statement" and "Node Pointer", separated by a `|` (pipe).

```
[[Are you sure, Robin?|NodeConfirm]]
```

The user should be forwarded to the **Node** labeled `NodeConfirm` when selecting the "*Are you sure, Robin?*" option.

8.5.2 Auto-forward Replies

In replies, you can leave out the statement, and just provide a **Node** Pointer, but you can only have one of these:

```
[[NodeConfirm]]
```

This should allow your user to go to the NodeConfirm node when selecting e.g. a default "Continue" button, or automatically after some time (up to your UI design). You can not have two of these options in the same Node, but you can mix them with Basic replies, like so:

```
Would you like me to sign you up?

[[Yes, please do so!|Confirm]]
[[No, let's not.|Cancel]]
[[UserInDoubt]]
```

In the example above, you could for example give the user some time to choose between the "*Yes, please do so!*" and "*No, let's not*" options, and after some time, automatically progress the dialogue to the `UserInDoubt` **Node**.

8.5.3 Input Replies

You can ask a user to provide various types of input using Input Reply options. The easiest way is to request some text input:

```
What is your first name?

[[None of your business Robin.|RobinInsulted]]
[[My name is <<input type="text" value="$userFirstName" min="2" max="30">>,
why do you ask?|RobinInputGiven]]
```

The general format of this statement is: (optional) `beforeText`, `inputStatement`, (optional) `afterText`. (optional) `min`, (optional) `max`.

That means that the following is the minimal example that is also valid:

```
What is your first name?

[[None of your business Robin.|RobinInsulted]]
[[<<input type="text" value="$userFirstName">>|RobinInputGiven]]
```

When a user chooses the Input Reply, the provided text is assigned to the value of the `$userFirstName` variable.

Very similar is numeric input:

```
[[I am <<input type="numeric" value="$userAge" min="0" max="120">> years
old.|RobinInputGiven]]
```

In both cases the `min` and `max` parameters are optional (you can have none, either, or both).

Furthermore, WOOL supports time input in hours and minutes:

```
[[I ate my breakfast at <<input type="time" value="$breakfastTime"
granularityMinutes="15" startTime="09:00" minTime="06:00" maxTime="12:00">>
this morning.|BreakfastTimeGiven]]
```

This results in the variable `$breakfastTime` being set to something like "07:45". There are four optional parameters:

- **`granularityMinutes`**: In the example this is 15, meaning that you can enter 00, 15, 30 or 45. You can use any value between 1 and 60. The default is 1.
- **`startTime`**: The time that the input widget should show initially. For example "09:00", but you can also write "now", or even a variable: "$breakfastTimeYesterday". If you leave it out, then the input widget will start empty.
- **`minTime`**: The minimum time that the user can enter. The default is "00:00".
- **`maxTime`**: The maximum time that the user can enter. The default is "23:59".

8.5.4 Replies with Setting Variables

Instead of setting a variable in a single set-**Statement**, you can also set a variable as part of a **Reply** option, like in the following example:

```
Do you prefer meat or fish?

[[Meat please.|NodeMeat|<<set $likesMeat = true>>]]
[[Fish for me.|NodeFish|<<set $likesFish = true>>]]
```

8.5.5 Replies with Actions

Just like you are able to link a set-**Statement** to a **Reply**, you can also add action-**Statement**s to replies, like so:

```
[[Please show me the recipes.|RecipesStart|<<action type="generic"
value="OPEN_RECIPE_BOOK">>]]
```

8.5.6 Replies linking to other dialogues

There's only so much you want to put into one WOOL dialogue definition before you start losing track (and/or sanity), so WOOL allows you to link between different dialogue definitions, like so:

```
What should we talk about now?

[[Know anything about cars?|CarsDialogue.Start]]
[[What about fishing?|FishingDialogue.Start]]
```

In this example, the first **Reply** option would take the user to the **Node** labeled "*Start*" of the Dialogue labeled "*CarsDialogue*". So, in this case, your application should be aware of a file named "CarsDialogue.wool". Note that you don't *have to* link to the "Start" **Node** of a dialogue script, and you can choose any valid **Node** name.

Note: currently, it's up to the application developer how to organize their WOOL script files, and thus how you deal with linking between dialogues. Support for "WOOL Projects" is currently in development, which should help streamline this process.

9 Annex B: WOOL Tutorial: Setting up WOOL for your Java project

This tutorial covers setting up your own Java program, including the WOOL Java library, and verifying that everything works correctly. This tutorial assumes a basic understanding of Java and will use the IntelliJ IDEA (version 2019.3.4), although you can use your IDE of choice.

9.1 Table of Contents

9.2 Before you get started

- This tutorial is using **IntelliJ IDEA**, which you can download from **JetBrains**.
- All source code used in this tutorial is available from the **WOOL GitHub** pages, specifically from the **wool-java-tutorials** repository.

9.3 Setting up the tools

Step 1: Create a new project in IntelliJ by selecting the "+ Create New Project" option from the start-up screen. Alternatively, select "File -> New -> Project…" from the IntelliJ menu bar.

Figure 27: Image 1 of the WOOL Tutorial – Setting up WOOL for your Java project.

Step 2: In the "New Project" dialogue box, Select "Gradle" on the left, and make sure to check the "Java" box from the "Additional Libraries and Frameworks" field. Then, hit "Next".

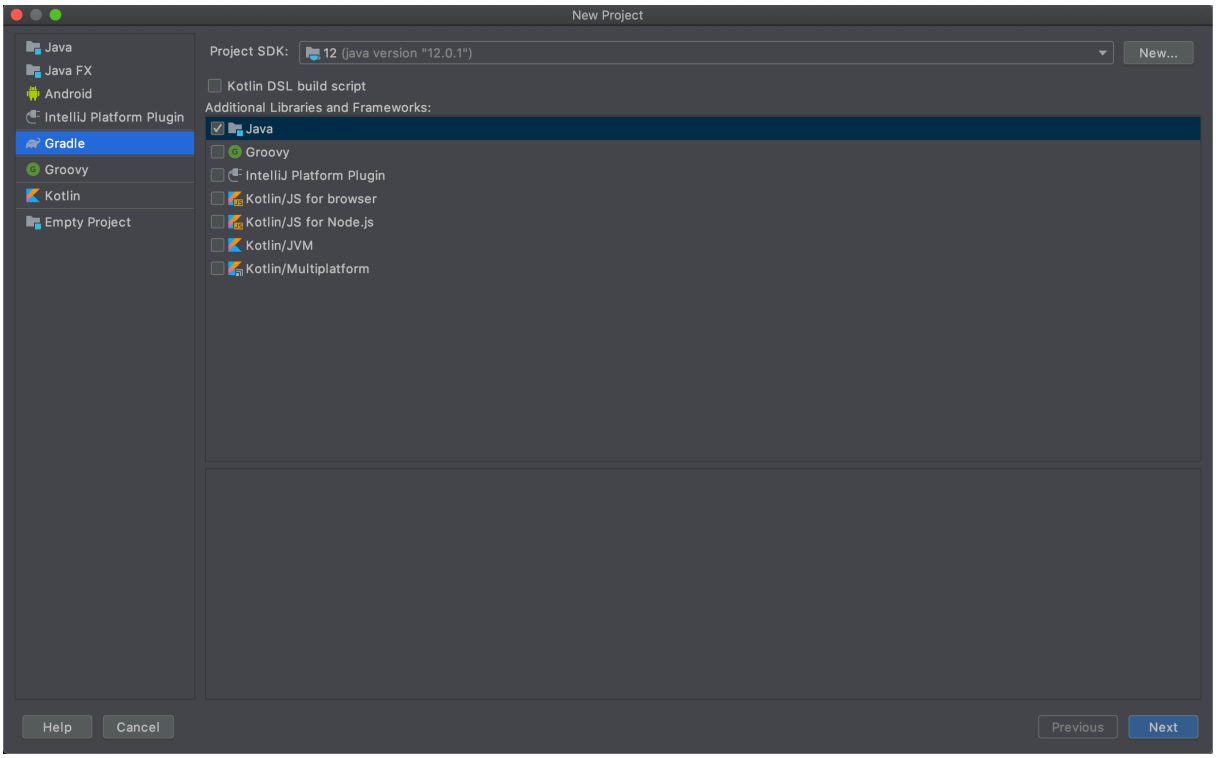


Figure 28: Image 2 of the WOOL Tutorial – Setting up WOOL for your Java project.

Step 3: Choose a name for your project (e.g. "WOOLBasicTutorial") and a location where you want to store your project files (e.g. under a GIT repository). Unfold the "Artifact Coordinates" option to reveal some additional options for your project setup. We're going to leave them as is for now. Then, hit "Finish".

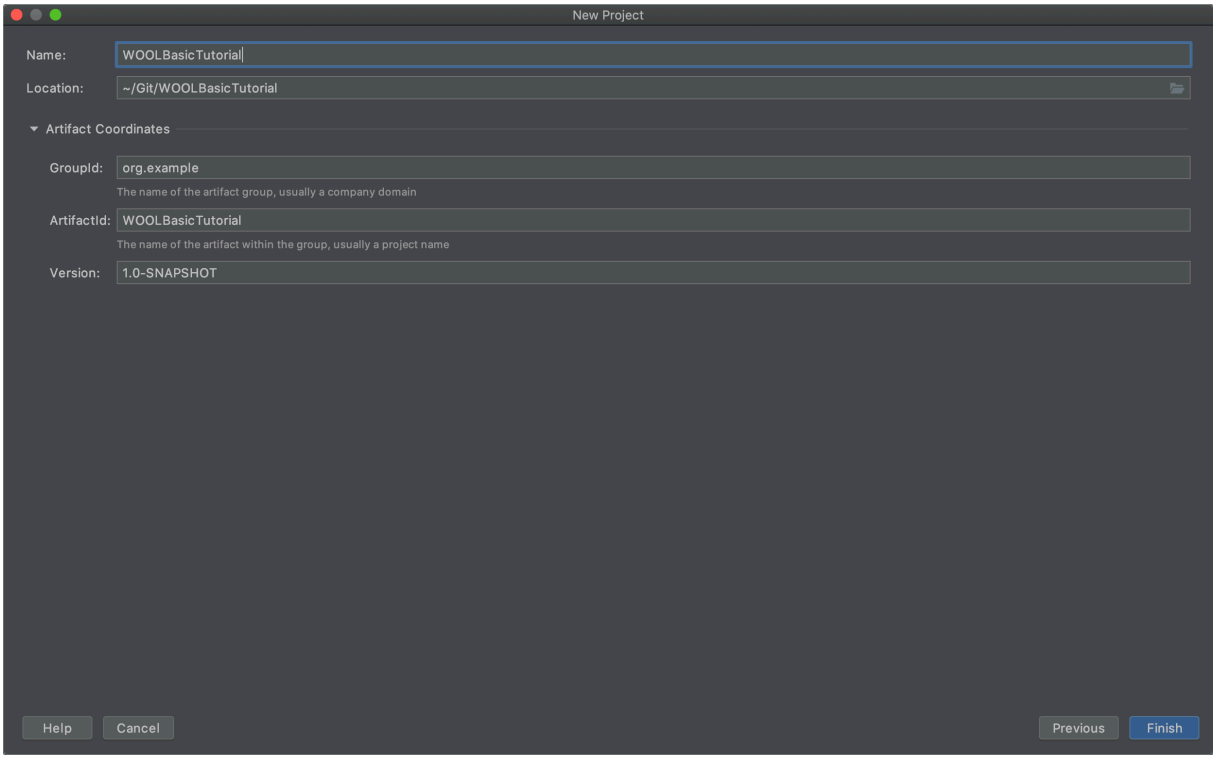


Figure 29: Image 3 of the WOOL Tutorial – Setting up WOOL for your Java project.

Step 4: IntelliJ will perform some basic setup operations and download the necessary gradle libraries. When finished your new empty workspace will look something like the figure below.

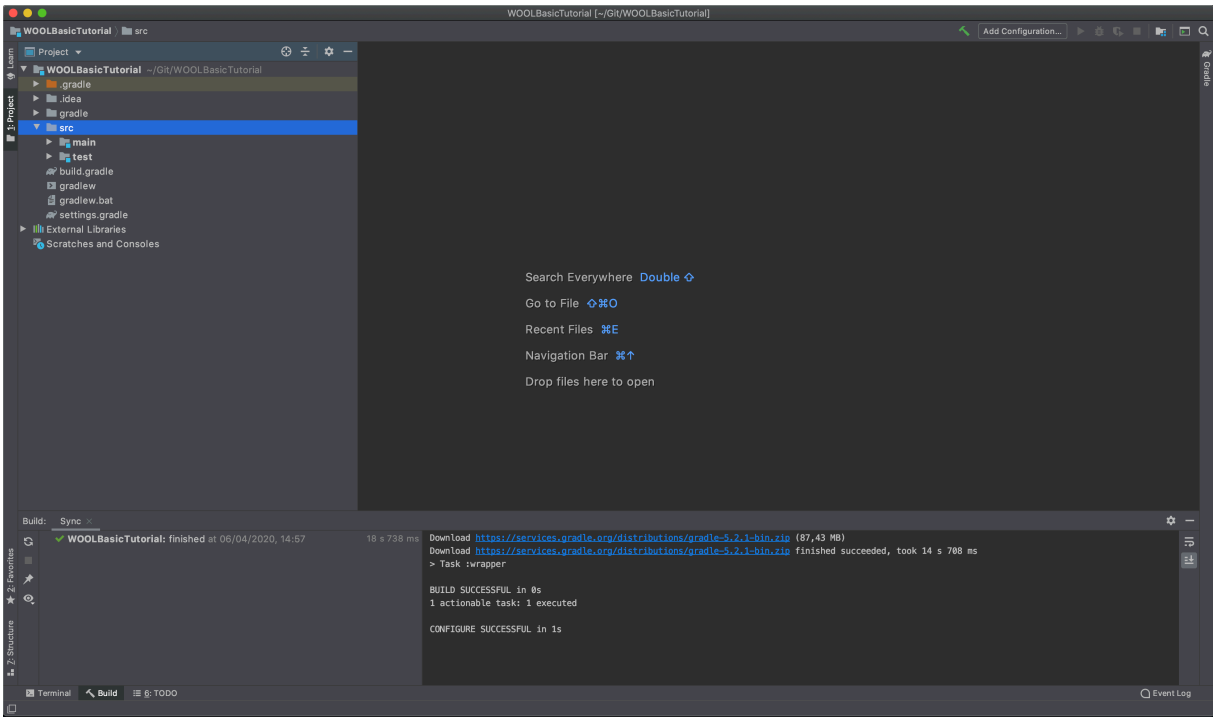


Figure 30: Image 4 of the WOOL Tutorial – Setting up WOOL for your Java project.

9.4 Importing the WOOL Java Library

To finish our setup, we need to add the WOOL Java library as a project dependency, so that we can start using it in our project. To do this, open the build.gradle script by double-clicking on it from the Project explorer.

Then, add the following line to the dependencies part of the Gradle build script:

```
compile 'eu.woolplatform:wool-core:2.0.0'
```

IntelliJ might warn you that “Gradle projects need to be imported” (in the bottom right corner). Go ahead and click “Import Changes” or “Enable Auto-Import” to always do this without notification in the future.

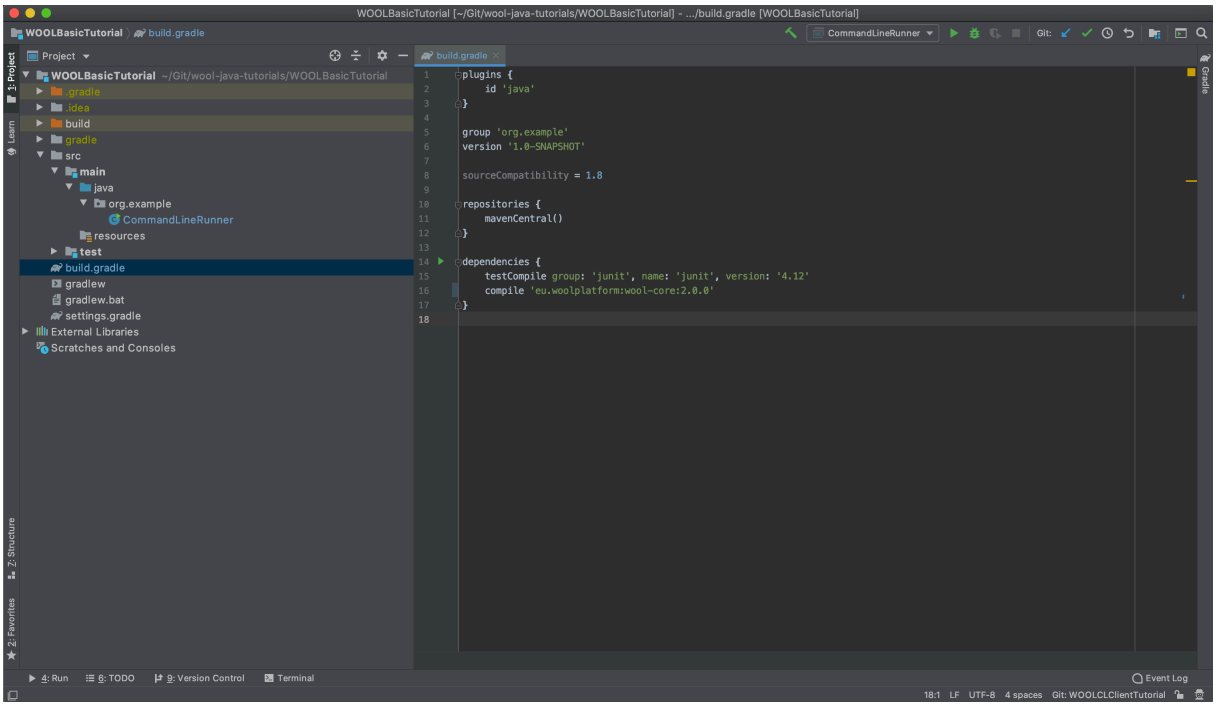


Figure 31: Image 5 of the WOOL Tutorial – Setting up WOOL for your Java project.

Afterwards, the build.gradle script should look like this:

```
plugins {
    id 'java'
}

group 'org.example'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
    compile 'eu.woolplatform:wool-core:2.0.0'
}
```

9.5 Creating your own class and testing the setup

Step 1: First let's create our own Java class that we will call the "CommandLineRunner". Find the src\main\java folder, right-click and select New -> Java Class. For the name we type "org.example.CommandLineRunner". IntelliJ will initialize this simple class for us in the org.example package.

Step 2: To test whether our project setup is working as expected, we will create a very simple constructor class and make a reference to a WOOL Library class, like so:

```
package org.example;

import eu.woolplatform.wool.model.WoolDialogue;
```

```
public class CommandLineRunner {

    public CommandLineRunner() {
        WoolDialogue wd = null;
    }
}
```

IntelliJ should provide auto-completion options for the WOOL* classes and automatically suggest the import of eu.woolplatform.wool.model.WoolDialogue. So far so good, let's do something interesting.

9.6 Command Line Runner Basics

First, we are going to read in an existing .wool script and see if we can show some information about this script on screen.

For this, we are going to need a function that we will call "analyzeWoolScript" that takes as input a fileName.

```
private void analyzeWoolScript(String fileName) {
    System.out.println("There should be a WOOL Script here: " + fileName);
}
```

Next, we need to call that function with the actual location of a .wool file, and since we are creating a "Command Line Runner", let's take that input from the command line. For that we will create a runnable function "main", that will ask the user for input and pass it to our newly created analyzeWoolScript() function:

```
public static void main (String[] args) {
    // create a scanner so we can read the command-line input
    Scanner scanner = new Scanner(System.in);

    //  Ask for a .wool script
    System.out.println("Path to WOOL Script: ");

    // Get the input as a String
    String woolScriptFile = scanner.next();

    // Initialize the CommandLineRunner and call analyzeWoolScript()
    CommandLineRunner clr = new CommandLineRunner();
    clr.analyzeWoolScript(woolScriptFile);
}
```

We now have a piece of software that we can run, so let's give it a try. Right-click the "CommandLineRunner" class file, and select "Run 'CommandLineRun.....main()'". You will see the output of the program asking for the Path to the WOOL Script. Type something, and see the results!

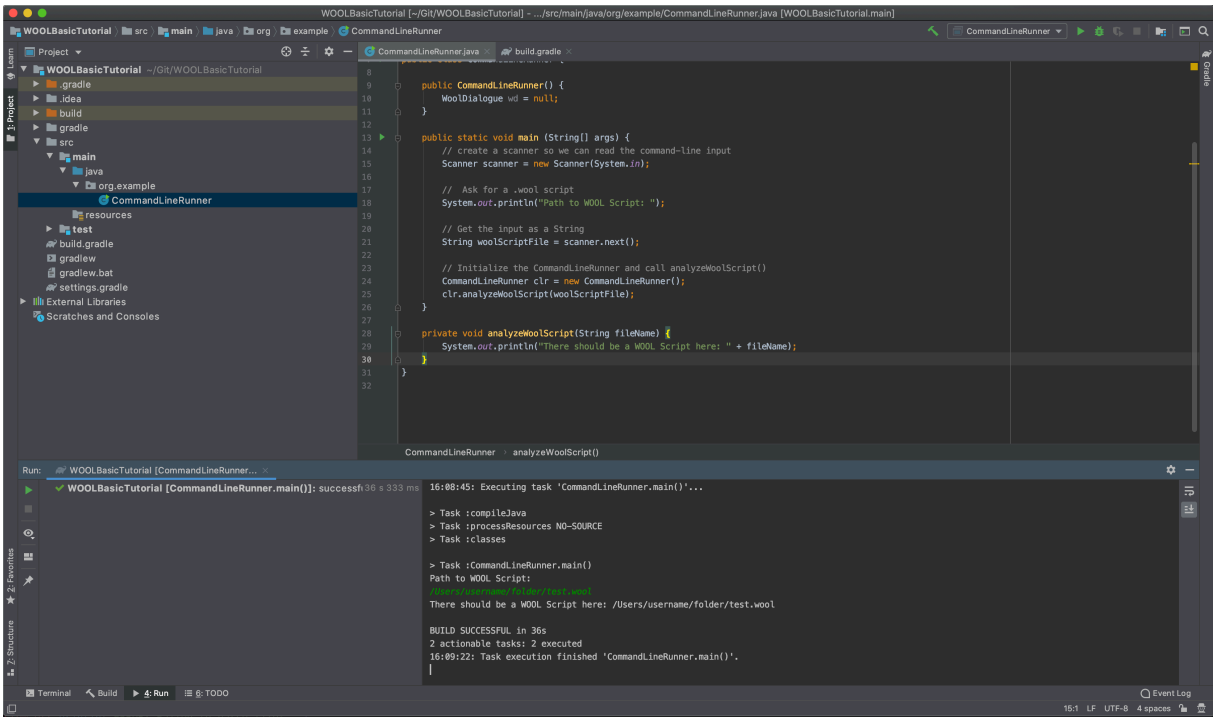


Figure 32: Image 6 of the WOOL Tutorial – Setting up WOOL for your Java project.

9.7 Reading and Parsing the WOOL Script

For the next step, we first need an actual .wool script. For this, we are going to use the example .wool scripts that come with the WOOL repository on GitHub. If you haven't already done so, go ahead and clone the wool repository to your local disk. Whatever your folder is in which you store your git projects, we're going to call that $GIT_DIR.

Now it's time to do something WOOL-related in our analyzeWoolScript() method. We are going to initialize a WoolParser for the given file, read it in, and print out some basic information about the WOOL script. See the code snippet below:

```
private void analyzeWoolScript(String fileName) {
   System.out.println("There should be a WOOL Script here: " + fileName);

   // First, create a File object from our fileName String
   File file = new File(fileName);
   if (!file.exists()) {
       System.err.println("ERROR: File not found: " + fileName);
       System.exit(1);
       return;
   }

   // Initialize a ReadResult where the results of the parse will be stored
  WoolParserResult parserResult;

   try {
       // Create a new WoolParser for the given file
       WoolParser parser = new WoolParser(file);

       // Parse the WOOL script and store the results in readResult
       parserResult = parser.readDialogue();

       // Retrieve the WoolDialogue representation from the readResult
       WoolDialogue woolDialogue = parserResult.getDialogue();
```

```
        // Output some basic information about the WOOL script
        System.out.println(woolDialogue.toString());
    } catch (IOException ex) {
        System.err.println("ERROR: Can't read file: " +
                file.getAbsolutePath() + ": " + ex.getMessage());
        System.exit(1);
        return;
    }

}
```

Now let's try to run this, and see what happens. Run the application again, and when asked, point to the basic.wool test dialogue that is located in the wool repository (see Figure below):

```
$GIT_DIR/wool/test-dialogues/basic.wool
```

(remember to replace $GIT_DIR with wherever your git repository is located)

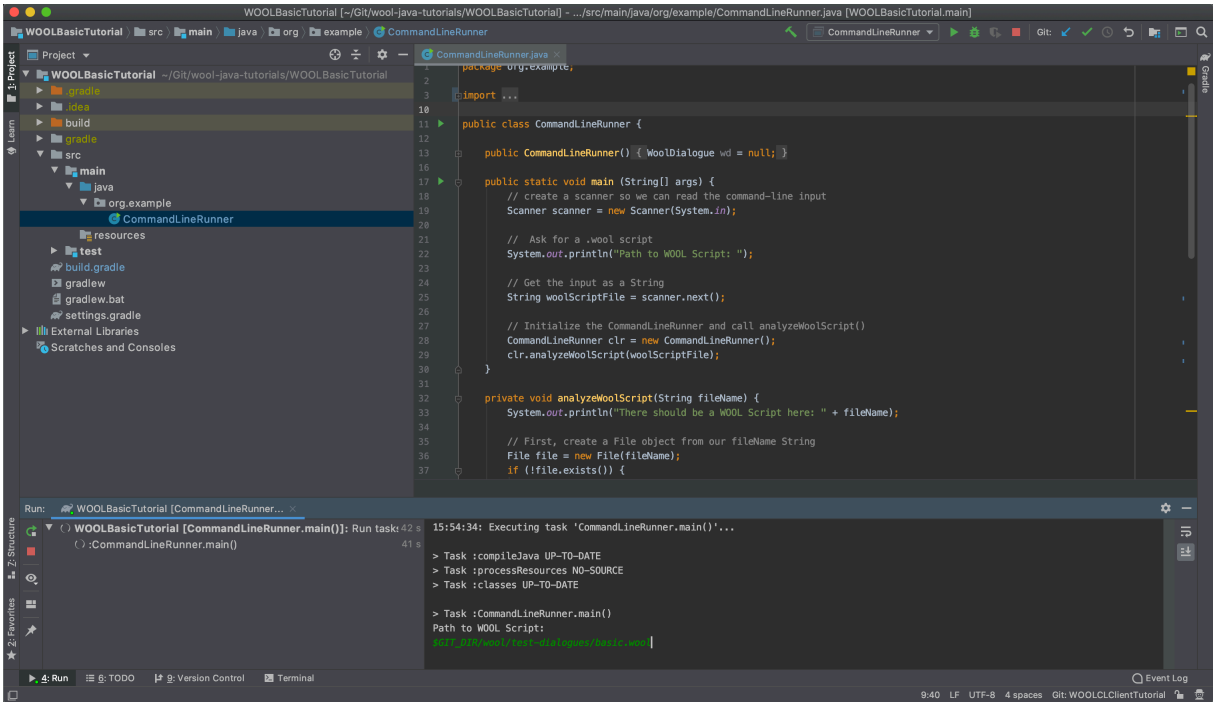


Figure 33: Image 7 of the WOOL Tutorial – Setting up WOOL for your Java project.

After pressing "RETURN", the application should output the following:

```
There should be a WOOL Script here: $GIT_DIR/wool/test-dialogues/basic.wool
Dialogue Name: basic
Number of Nodes: 9

Speakers present (1):
  - Bob
Dialogues referenced (0):
Variables needed (0):
Variables written (0):
```

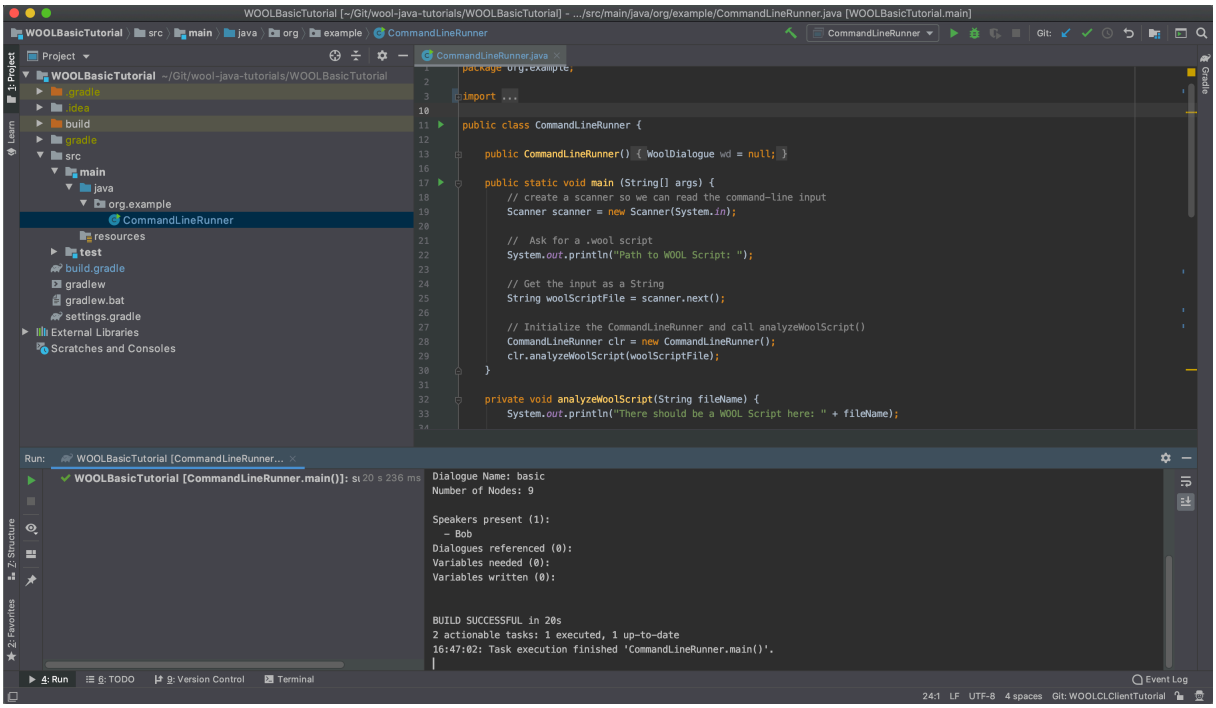


Figure 34: Image 8 of the WOOL Tutorial – Setting up WOOL for your Java project.

Et voila! You have successfully set up your Java environment to start using WOOL in your applications.

10 Annex C: WOOL Editor Tutorial: How to make a standalone interactive fiction game.

This tutorial describes how to make a simple standalone interactive fiction game. You can use **the web-based Wool editor**. NOTE: for more complex applications involving a Java server or a custom dialogue client, the desktop Wool editor is recommended. It has file management and multi-dialogue support.

10.1 Using the editor

You can find the web-based editor here.

The editor starts with a single Start node. You can click on the node and start editing it.

Figure 35: Image 1 of the WOOL Tutorial – How to make a standalone interactive fiction game.

First, you have to fill in the Speaker field. This is used when in a dialogue with a character in your story. If you do not want a dialogue, but want to describe a location with associated actions from the player, fill in Narrator.

The Wool language supports multiple choice actions, text input, variables, and conditionals. For a full specification, see **the Github Wiki**.

The editor also supports HTML tags in the text, for example you can use <p> for a paragraph break or ... for boldface text. Using File->Open, you can open Wool files from your local drive. Download the file below and open it.

Example Wool file: a simple game called "The Gnome".

Your editor should look like this:

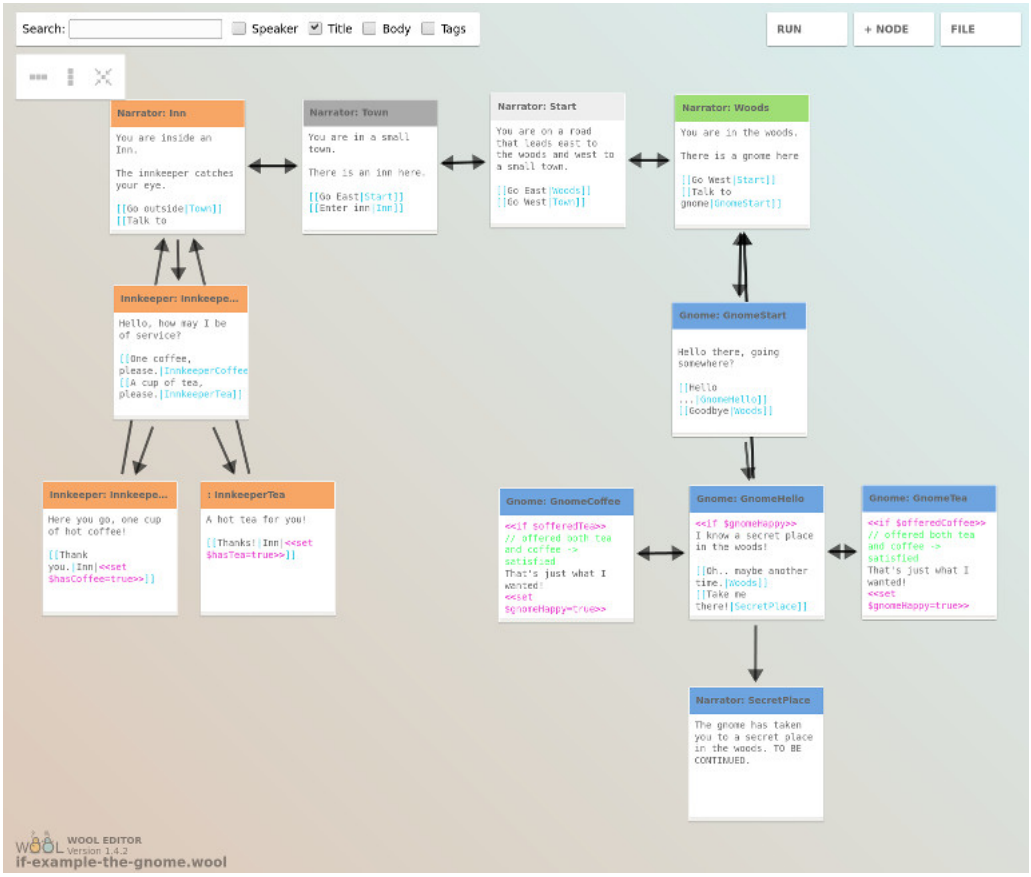


Figure 36: Image 2 of the WOOL Tutorial – How to make a standalone interactive fiction game.

Note that the nodes all have different colours. Change the colours using the arrows that appear when hovering over a node. Each colour can be mapped to a different background, once we are in the dialogue viewer.

Now, let's look at some code. Here is the start node:

```
You are on a road that leads east to the woods and west to a
small town.

[[Go East|Woods]]
[[Go West|Town]]
```

Note that the speaker is Narrator, which indicates this is a location and not a dialogue. The lines between double brackets indicate options for the player. The left hand side is the text to show, the right hand side is the name of a node. Other, more complex options are also possible.

Let's look at the Innkeeper dialogue.

```
Hello, how may I be of service?

[[One coffee, please.|InnkeeperCoffee]]
[[A cup of tea, please.|InnkeeperTea]]
```

Format is exactly the same as the Start node, but because we filled in a speaker, we now get a dialogue with an avatar. Here's the InnkeeperCoffee node:

```
Here you go, one cup of hot coffee!

[[Thank you.|Inn|<<set $hasCoffee=true>>]]
```

Here, we set a variable $hasCoffee as a side effect to an option. Now, we can do something with the variable later on. Note that variables always start with a "$". This makes it possible to use them in the text, like in "Hello, $playerName".

In our game, we encounter a gnome. Here is part of the dialogue:

```
All alone in the woods? Do you have something for me?

[[I don't know what you are talking about!|Woods]]

<<if $hasCoffee>>
    [[I have a coffee.|GnomeCoffee]]
<<endif>>

<<if $hasTea>>
    [[I have tea.|GnomeTea]]
<<endif>>
```

You can surround any part of the code with an if...endif condition. Else and elseif are also supported.

Now, let's try and run the game. Press "Run".

Figure 37: Image 3 of the WOOL Tutorial – How to make a standalone interactive fiction game.

Within the Wool viewer, you can edit the graphical appearance of your game. The control panel at the bottom left can be used to change the current background, and avatar if there is a speaker defined. Use

"+" and "-" to cycle through the options. A number of pre-sets are defined. Use "URL" to enter a new image URL. The trashcan button can be used to delete previously entered image URLs. The viewer will remember your choice of avatar for each speaker, and your choice of background for each node colour.

When you are happy with the looks, you can export your game using the "Get URL" button. You will see a (huge) URL that encodes both your source code and the graphical configuration. Note that some browsers have a 4Kb limit on URL size (Internet Explorer) though bigger URLs will work fine in Chrome or Firefox.

There are also buttons for going back to the editor, and editing the current node. Note it's also possible to edit the dialogue and then continue it from where it left off.

Within the editor you can save your Wool file, but saving the graphics is still a bit primitive. You can use the result of "Get URL" to save the graphical configuration. In particular the URL parameter "config=....." can be copy/pasted into the viewer URL to copy it into another game.

Another useful Wool viewer URL parameter is "resetconfig". If you add "&resetconfig=true" to the end of the URL, the graphical configuration is reset to its default state.

Finally, here is the URL to the Gnome game with some custom graphics: **Click here to Play "The Gnome"**.

Acknowledgements

The Council of Coaches project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement #769553. This result only reflects the author's view and the EU is not responsible for any use that may be made of the information it contains.

Headings and titles in this document, as well as the Council of Coaches logo use the Comfortaa font, designed by Johan Aakerlund and Cyreal and licensed under the Open Font License[2].

Additional text in this document uses the Roboto font, designed by Christian Robertson and licensed under the Apache License, Version 2.0[3].

The Council of Coaches logo and Blobmen graphics were *drawn freely* in Inkscape, licensed under the GNU General Public License[4].

[2] Open Font License: http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=OFL_web
[3] Apache License, Version 2.0: http://www.apache.org/licenses/LICENSE-2.0
[4] Inkscape License Information: https://inkscape.org/about/license/