From Mental Network Models to Virtualisation by Avatars: A First Software Implementation

de Jong, Frank C.; Eler, Edgar; Rass, Lars; Treur, Roy Michael; Treur, Jan; Koole, Sander

published in
Biologically Inspired Cognitive Architectures 2021
2022

DOI (link to publisher)
10.1007/978-3-030-96993-6_7

document version
Publisher's PDF, also known as Version of record

From Mental Network Models to Virtualisation by Avatars: A First Software Implementation

Frank de Jong[1], Edgar Eler[1,2], Lars Rass[1], Roy M. Treur[1], Jan Treur[1(✉)], and Sander L. Koole[2]

[1] Social AI Group, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
contact@frankdejong.software, {e.eler,j.treur}@vu.nl, l.f.science@b-rass.de

[2] Department of Clinical Psychology, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
s.l.koole@vu.nl

Abstract. Mental processes in the brain and body are often modelled and simulated by causal models according to a dynamical systems perspective. These causal models can be adapted to fit natural human processes and describe specific human traits. Patterns or time series generated by simulations are often displayed in the form of graphs of activation levels of mental states against time. However, from these generated time series generated by such a model, using avatars virtual agents can be constructed which express these patterns. Such forms of virtualisation can be used in therapy or coaching sessions to help clients gain insight into their behaviour and understand their functioning better. This paper describes part of a contribution to the virtualisation project CoSiHuman to achieve this, focusing here on first steps toward development of a software environment supporting this path. The presented approach utilises the Unity game engine, which includes the creation of several human-like avatars with the ability to express several human emotions. Several programming libraries were created to facilitate easy application of simulation data from causal models to obtain virtual agents based on avatars. These dynamic libraries were made to be easy to expand and be used by other researchers for future occasions. The approach was successfully applied to simulation based on an example causal model in the context of a couple's therapy showing its utility. Finally, the software was evaluated by two experienced software developers from industry and found to be good, well-documented and easily extendable.

1 Introduction

Simulating human mental processes has been the subject of much research in the past decade; e.g., (Treur 2020). These simulations use human-like computational causal models based on causal pathways in the brain and body. These causal models contain connections between mental states and how they interact and behave together. Mental processes are based on complex networks, often with cyclical connections, between several mental states (Kim 1996). Causal models can be adapted to fit natural mental processes and specific human traits. From simulations by such models, virtual agents based on avatars

V. V. Klimov and D. J. Kelley (Eds.): BICA 2021, SCI 1032, pp. 75–88, 2022.
https://doi.org/10.1007/978-3-030-96993-6_7

can be created, which express the generated patterns of mental states over time. This approach can be used in coaching or therapy sessions as a way for people to understand their functioning better. Since this is all virtual, the scenarios can be flexible and convey complicated behaviours involving multiple people. Moreover, these virtual agents can also be used in games or education. They can help to provide insight into human mental processes.

This process of virtualisation involves creating digital, human-like avatars. These avatars act as a medium for expressing human-like attributes using the patterns generated by simulations from the computational models. This paper focuses on such virtualisation by digital avatars, how they are created, and how they can be applied to causal models to express human-like patterns and underlying traits. Humans observe the avatars depicting some natural process to gain insight into their own behaviour and underlying mental processes. An approach is explored which uses existing software to build virtual agents based on avatars and causal models in a systematic manner. Thus a reusable software environment can be obtained which makes it easier to achieve virtualisation of any causal mental model and its simulations, so that this does not need to be done from scratch all the time. The main software for virtualisation is the Unity software. As an ilustrative example, two virtual avatars are created and imported as assets into a Unity project. Libraries were built that allow the avatars to be controlled for a wide variety of potential applications.

The above considerations have led to a methodology based on adaptive mental causal network models and avatars expressing their simulation; see the CoSiHuman project described in (Treur et al. 2021) or URL https://www.researchgate.net/project/CoSiHuman-Cooperative-Simulated-Human. The shared mental networks used for mental processes is a basic similarity between natural humans and the obtained artificial humans.

2 The Example Causal Mental Network Model Used

In the network-oriented modeling approach based on temporal-causal networks described in (Treur 2020), a network structure is defined by *network characteristics* for connectivity, aggregation and timing. Network nodes have activation values that change over time: they serve as state variables and (like in Philosophy of Mind) are also called (mental) states. More specifically, a (temporal-causal) network model is characterised by (here X and Y denote nodes of the network, also called states):

- *Connectivity characteristics*
- Connections from a state X to a state Y and their weights $\boldsymbol{\omega}_{X,Y}$
- *Aggregation characteristics*

  For any state Y, some combination function $\mathbf{c}_Y(..)$ defines the aggregation that is applied to the impacts $\boldsymbol{\omega}_{X,Y}X(t)$ on Y from its incoming connections from states X
- *Timing characteristics*

  Each state Y has a speed factor $\boldsymbol{\eta}_Y$ defining how fast it changes for given impact.

Such states are depicted in Fig. 1 by the small ovals and the causal relations between them (connections in the causal network) by arrows.

The following difference (or differential) equations that are used for simulation purposes and also for analysis of temporal-causal networks incorporate these network characteristics $\boldsymbol{\omega}_{X,Y}$, $\mathbf{c}_Y(..)$, $\boldsymbol{\eta}_Y$ in a standard numerical format:

$$Y(t+\Delta t) = Y(t) + \eta_Y\left[\mathrm{c}_Y\left(\boldsymbol{\omega}_{X_1,Y}X_1(t), \ldots, \boldsymbol{\omega}_{X_k,Y}X_k(t)\right) - Y(t)\right]\Delta t \qquad (1)$$

where X_1 to X_k are the states from which state Y has incoming connections. Such equations are hidden in the dedicated software environment; see (Treur 2020), Ch 9. There are many different approaches possible to address the issue of aggregating multiple impacts by combination functions. Therefore, for this aggregation a combination function library with a number of basic combination functions (currently more than 50) is available, while also own-defined functions can be added. Examples of basic combination functions from this library can be found in Table 1.

Table 1. Basic combination functions from the library used in the presented model

	Notation	Formula	Parameters
Stepmod	$\mathbf{stepmod}_{\boldsymbol{\rho},\boldsymbol{\delta}}(V)$	0 if time $t < \boldsymbol{\delta} \bmod \boldsymbol{\rho}$, else 1	Repetition parameter $\boldsymbol{\rho}$ Duration parameter $\boldsymbol{\delta}$
Advanced logistic sum	$\mathbf{alogistic}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(V_1, \ldots, V_k)$	$[\frac{1}{1+\mathrm{e}^{-\boldsymbol{\sigma}(V_1+\cdots+V_k-\boldsymbol{\tau})}} - \frac{1}{1+\mathrm{e}^{\boldsymbol{\sigma}\boldsymbol{\tau}}}](1+\mathrm{e}^{-\boldsymbol{\sigma}\boldsymbol{\tau}})$	Steepness parameter $\boldsymbol{\sigma} > 0$ Threshold parameter $\boldsymbol{\tau}$

As an example, consider a couple therapy for partners A and B, where partner A has aggression issues and being-in-control issues. For example, the following script can be generated by causal models for emotions and emotion regulation; e.g., (Essau et al. 2017). The aim for using a virtualised version of this script can be for the couple to gain more insight in interaction patterns relating to anger and fear regulation.

- Person A shows some level of anger; no emotion regulation
- Person B (seeing that) gets a stressful emotion
- Person B's gaze goes away, not at A anymore (fear emotion regulation: attention deployment)
- Person A (seeing that) shows more anger; no emotion regulation
- Person A makes a threatening gesture
- Person B (seeing that) gets a stronger stressful emotion
- Person B walks out of the room (fear emotion regulation: situation modification)
- Person A (seeing that) shows still more anger; no emotion regulation
- Person A breaks some valuable thing in the room

These are interaction processes that can well be modeled well by a causal modeling approach, where Person A has poor anger emotion regulation incorporated together with

a dependency on feeling fully in control of situations and Person B well-functioning (but for A undesirable) emotion regulation based on attentional deployment and situation modification. This has been modeled in a network-oriented manner according to (Treur 2020) as shown in Fig. 1; for an explanation of all states, see Table 2.

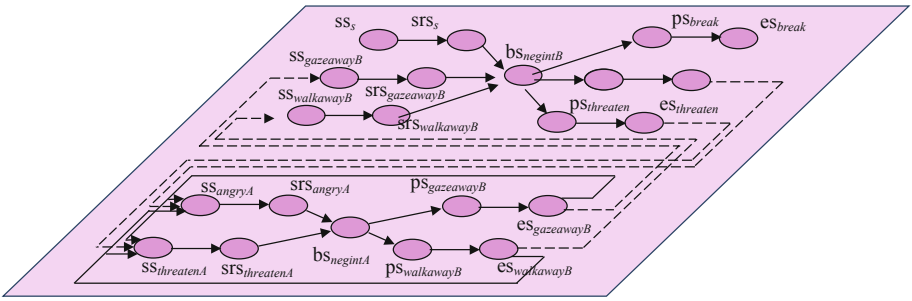

Fig. 1. The connectivity of the example causal mental network model

An example simulation of this network model is shown in Fig. 2. The full specification by role matrices is included in the Appendix.

Table 2. Overview of the states in the example causal mental network model

State nr	State name	Explanation
X_1	ss_s	Sensor state for stimulus *s*
X_2	$ss_{gazeawayB}$	Sensor state for seeing *gazeawayB*
X_3	$ss_{walkawayB}$	Sensor state for seeing *walkawayB*
X_4	srs_s	Sensory representation state for stimulus *s*
X_5	$srs_{gazeawayB}$	Sensory representation state for *gazeawayB*
X_6	$srs_{walkawayB}$	Sensory representation state for *walkawayB*
X_7	$bs_{negintB}$	A's negative interpretation of B
X_8	ps_{break}	Preparation state for breaking
X_9	ps_{anger}	Preparation state for anger
X_{10}	$ps_{threaten}$	Preparation state for threatening B
X_{11}	es_{break}	Execution state for breaking
X_{12}	es_{anger}	Execution (expression) state for anger
X_{13}	$es_{threaten}$	Execution state for threatening B

(*continued*)

Table 2. (*continued*)

State nr	State name	Explanation
X_{14}	ss_{angryA}	Sensor state for seeing angry A
X_{15}	$ss_{threatenA}$	Sensor state for seeing threatening A
X_{16}	srs_{angryA}	Sensory representation for seeing angry A
X_{17}	$srs_{threatenA}$	Sensory representation for seeing threatening A
X_{18}	$bs_{negintA}$	B's negative interpretation of A
X_{19}	$ps_{gazeaway}$	Preparation state for gaze away
X_{20}	$ps_{walkaway}$	Preparation state for walking out of the room
X_{21}	$es_{gazeaway}$	Execution state for gaze away
X_{22}	$es_{walkaway}$	Execution state for walking out of the room

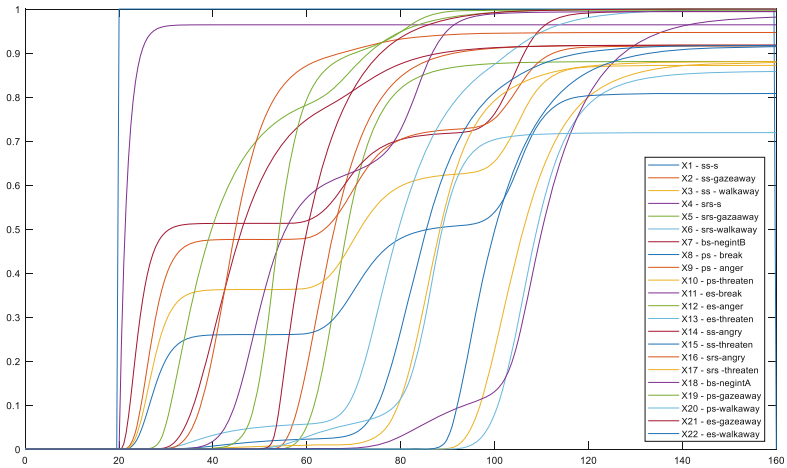

Fig. 2. Simulation outcome for the example model

3 The Setup of the Software Environment for Virtualisation

Once the avatars are virtualised, they are meant to be in interaction with humans, so that humans can obtain more insight in the mental processes. When natural humans observe avatars, the avatars must elicit empathy and a connection from the observer. Empathy, or the ability to relate to another person's feelings or emotions (Singer and Tusche 2014), is suggested to be triggered automatically (Preston and de Waal 2002). Empathy allows the observer to connect with the behaviour of the avatars and helps to gain insight into their behaviour. One element of accomplishing empathy (in addition to the notion of self-other distinction) is through emotion contagion, which is when an individual observes a behavioural change and reflexively produces the same behaviour or emotion (Burgos-Robles et al. 2019). The avatars should be applicable to simulation

data from several different causal models to provide insight into several behaviours. To allow the avatars to express several unique human-like traits, they must have the capacity for many human functions.

The avatars must express the six basic emotions: anger, disgust, fear, happiness, sadness and surprise, from Ekman's Theory of Basic Emotions (Miller 2016). Although Ekman later theorised that there are several other basic emotions beyond the initial six (Ekman and Cordaro 2011). This work reported in the current paper focused on the initial basic emotions due to the scope of the research but leaves the door open for more emotions to be implemented and integrated easily. From Ekman's 'Unmasking the Face' (Ekman and Friesen 2003), emotions can be encoded using the FACSHuman plugin for Makehuman (Gilbert et al. 2021) for each basic emotion. This encoding is achieved using the Facial Action Coding System (FACS) developed by Ekman, Hager and Friesen (Ekman et al. 2002). Each emotion is split into several 'Action Units', which are applied separately. Splitting ensures that each group of Action Units can be triggered independently. Triggering different parts together can blend emotions, e.g., fear in addition to surprise, by triggering the upper Action Units for fear and the bottom Action Units for the surprise emotion (Ekman and Friesen 2003).

Furthermore, the avatars must be able to walk, run, strafe, and turn. The approach can be expanded to support even more complex virtual behaviours. For example, virtual avatars can drive a car based on a causal model for road rage issues.

The Makehuman application is an open-source tool 'designed to simplify the creation of virtual humans using a Graphical User Interface' (Makehumancommunity.org 2016). This software is ideal since it abstracts from the complexity of 3D modelling and allows for the creation of realistic virtual human-like avatars. Realistic and relatable looking avatars are essential, given that a higher level of empathy can be obtained based on appearance and similarity to the avatar (Hall and Woods 2005). The creation of the avatar is very straightforward, with sliders for the avatar's visual features, i.e., gender, age, weight, height, race, etc. The approach also has many plugins that allow for additional features and customisations to be added. A plugin to the Makehuman application (Gilbert et al. 2021) is utilised for encoding facial expressions built on the FACS (Ekman et al. 2002).

The avatar from Makehuman is exported into the Blender software (Blender Foundation 2018), an open-source 3D creation suite. Support for the entire 3D pipeline is included, allowing the avatars to be animated, modelled, rendered, etc. A feature called blend shapes is used, enabling the shape to be deformed into a new shape gradually by scaling, transforming or rotating any of its vertices (Blender Foundation 2021). The blend shape can be applied gradually, so vertices are deformed over time or to a limited value. This technique is often used for animation.

At first, a base avatar is exported from Makehuman into Blender, which acts as a neutral shape. In Makehuman, the FACS system is used to export the same avatar with a different facial expression. These two avatars are then merged, and the difference between the base avatar and the morphed avatar is converted into a blend shape. Now, triggering this blend shape morphs the base avatar into the avatar with the facial expression. Animations are collected from Mixamo (MCV 2021), which is part of the Adobe creative cloud. These are pre-made and freely available animations that can be applied

to custom uploaded avatars. Using existing animations removes the tedious and often challenging task of creating animations.

The Unity game engine, or Unity software, is the final software used in the work reported in this paper. The avatar can be programmed to express human traits in the engine. The data used to express the human traits is collected from simulation data from the causal models. Unity is a game engine created by Unity Technologies and allows 3D and 2D games to be created. The engine is also used in the film, architecture, engineering and construction industries. Importing the blender objects is very well supported. Unity takes care of almost all parts of the virtualisation, from rendering the avatars, world creation, controlling the camera, sound, and any other component used to show the avatars. Unity is also a very powerful engine because it supports almost any platform easily. In the scope of the work reported in this paper, it allows easy cross-platform access to the virtualised avatars. Moreover, Unity is a good fit in avatar virtualisation as it already has many features to optimise performance and quality.

4 Creating Avatars

Avatars are first created in the Makehuman application. More specifically, the avatar's features are specified, e.g., age, gender, race, weight, the shape of the face, etc. Other features like clothing, hairstyle, and pose are also selected. In this first creation step, almost all visual attributes are specified. The avatar can be made to look human-like to the extent that the makehuman-project allows, which is a limitation. Once this base avatar is created, a so-called skeleton must be added, which will be used to add animations later on. Skeleton animation or rigging, in which the mesh (the surface of the avatar) is connected to the avatar's bones. Rotating, scaling, or translating these bones deforms the mesh and combining these deformations with several bones can create an animation. Makehuman provides several skeletons, one of them being 'advanced skeletons'. Advanced skeletons are skeletons that allow for complete control of all parts of the avatar. This paper's scope did not require precise control of the hands and feet bones; however, adding these to new avatars is also possible and can be done in future development. A more straightforward skeleton is used, which is also less taxing on the game engine. The final step is to export the avatar to an FBX format, imported into Blender.

In Blender, the facial shape keys (blend shapes) are added. This part consists of importing the base avatar first with a neutral expression. Once the Action Units, according to the theoretical framework, are applied to the avatar, it is exported to Blender. When both avatars are imported, one with an applied Action Unit and the other base avatar, each avatar component is merged to form several blend shapes. The avatar is split up into the body mesh, which contains the arms, legs, and face, the eyebrows mesh, the eyes mesh, the tongue mesh, the teeth mesh, the clothing mesh, and the hair mesh. Depending on the Action Units applied, not all components are deformed, and therefore, the blend shape is only added to these morphed components. For example, moving only the eyebrows deforms the eyebrows component, so only a simple blend shape is added. However, moving the mouth, body, tongue, and teeth are all deformed, so a blend shape is added for each of those components. This process is repeated for each of the six basic emotions and each individual Action Unit specified in 'Unmasking the face' (Ekman and Friesen 2003); see Fig. 3.

Fig. 3. Full-face fear expression encoded from Fig. 22 in 'Unmasking the Face' (Ekman and Friesen 2003) into Blender using FACSPlugin.

Adding blend shapes must be repeated for every new avatar created. The FACSHuman plugin allows each Action Unit to be saved, so the Action Units only need to be loaded each time a blend shape is added. This repetitive process also opens the door for a potential automation program that could repeat this process several times. Blender also has a scripting API, making avatar creation straightforward, employing an easy-to-use Graphical User Interface (GUI).

Animations allow the avatars to come to life by simulating movements like walking and running and other actions like throwing an object, hand gestures, and most other conceivable actions. Animating is a critical component to make the avatar human-like and allows the humans to relate and empathise better. However, a time-consuming obstacle is animating each bone in the avatar's skeletons. Professional animators are likely required to make these actions look believable. Animations are imported from Mixamo (MCV 2021), a library of existing animations, which saves time.

Once the avatar has animations and blend shapes, it can be imported into the Unity game engine, supporting all Blender features. Unity has a concept called a 'scene' where game objects can be added. Any object in Unity is a game object, from cameras, lights, characters, special effects, and anything else. Each game object can also have several child game objects, which can give it more properties. The avatars are created as a Prefab Variant, a special Prefab game object with stored properties. Typically, a game object is added to the scene, and its properties are then configured. A Prefab is a game object with the properties already defined. If the scene requires two or more of the same game objects with the same properties, a Prefab is helpful as the properties are already defined. When many of these game objects are in the scene, and the properties need to be edited, you do not need to edit each game object but just the Prefab. Editing the Prefab applies all the changes to each game object in the scene. A Prefab Variant inherits the properties of the Prefab but can override any number of properties. In the context of avatar virtualisation, a new Prefab Variant can be created for each different scenario. This way, all its properties are inherited and do not need to be added for each scenario. Instead, each scenario can create a variant and only change the required properties, saving a lot of time. The Prefab Variant for each avatar contains a 'Rigidbody', responsible for the game object to be affected by gravity, and Animator Component, allowing the animations to be added to the avatar. The variant also contains a Box Collider game object attached to both feet

of the avatar, which prevent it from falling through other objects and a script called the Avatar Emotion Controller, which controls the expression of emotions of the avatar. The use of the Prefab makes it easy for the avatar to be used in multiple different scenes with little to no setup required.

Now, the avatars are imported into the engine as assets. There must also be a way for the avatars to be controlled and programmed. A dynamic library is created, allowing the avatars to be programmed easily. Several scripts were made and organised into C# namespaces, a programming concept where several different functionalities are grouped. Only relevant classes are exposed in these namespaces. This way, complex implementation details are not revealed, keeping the usage simple. The 'FacialExpressions' namespace is responsible for allowing an avatar to show a facial expression or emotion. A single class called 'EmotionController' is made available in this namespace, which can be added to a script game object. The script object is a piece of code that can modify a component's properties, create components, trigger events, and respond to user input. Since the 'EmotionController' class must be attached to an avatar, a script called 'AvatarEmotionController' is created in which an 'EmotionController' class is instantiated. The 'EmotionController' class makes a method available in which the emotion name is given and the intensity. It will then make the avatar's face express that given emotion. This class uses a hidden class that is responsible for parsing the blend shapes in the following manner. Each avatar's blend shapes have the name of an emotion, e.g., angry, sadness, etc. A dash separates this and then the name of a facial area. Another dash separates the last part of the name, which is the so-called variation. Each blend shape is grouped by the initial name, representing the category and name of emotion. This is followed by a sub-category, the more specific part of the face involved. This is followed by the variation value, where 'a' is zero, and each letter next in the alphabet increments it by one. After these are parsed, it can be controlled by setting the main category, the emotion name, followed by an intensity value. All the blend shapes with an emotion name are collected, and each is then set to the intensity value. This way, all the corresponding blend shapes are expressed uniformly. However, this is also a hurdle, as some emotions might need to be expressed in only one part of the face. Furthermore, the game engine must set the intensity gradually and not in a single frame. This can be done by calling a coroutine, a loop that can start and then wait with executing for some time before continuing as if it never stopped. This way, every half a second, one iteration can be completed. This is ideal for setting the emotion intensity gradually to be a smooth transition from neutral to any emotion and from any emotion to neutral. The time for an emotion to get set can be specified as well.

Another class in the 'FacialExpressions' namespace is the 'EyeMovementController', which acts similarly to the emotion controller regarding the avatar's eye gaze position. It combines four blend shapes, two for moving the eye left and right and the other two for moving the eye up and down. The moving is abstracted to a single function that allows the position of the eyes to be moved to an x and y position.

Now that the avatars can express facial expressions and move the eyes, there also needs to be a way for the avatars to move. The avatar could walk away, run away, throw objects, turn around, etc. A lot of the required actions these avatars perform have some things in common. Namely, the implementation requires access to the avatar's Animator,

RigidBody, and Transform game objects. The Transform object represents where in the world this object is. An abstract class is created, which is implemented for some actions that the avatar needs to perform. This class ensures that the same function can be called on any class that inherits this abstract class. This function is called "trigger" and, as its name suggests, triggers the action performed. In any action, force can be applied to the RigidBody, an animation can be activated or deactivated, and the avatar can be translated or rotated relative to the world. This standard implementation also ensures that the actions are all triggered using a coroutine. This ensures the action isn't performed in a single frame but however long or short as required.

For the avatars to be applied to simulation data from a causal model, a scene must first be set up. In this scene, the avatars required can be added along with any other components that make up the world, cameras, lights, etc. The scene can be different for each model and is not addressed by the work described in the current paper. However, each scenario requires an interface between the avatars and the scenario. The easiest way to accomplish this is to create an empty game object, an invisible object. A script is then added to this game object; when run, it activates. It also contains all the processes specific to that model, i.e. reading the simulation data, initialising the references to the avatars in the scene, and applying the data to the scene over time.

The approach is set up in the Unity software. The components created include avatar A and B, which are created and then imported into Unity. These avatars are general and reusable in several different virtualisation scenarios. The libraries created for the avatars include the Emotion Controller, Eye Movement Controller, Actions and the 'ModelData' script. However, several components need to be created to connect the virtualisation to simulation data from a model. Namely, a specific script responsible for controlling the avatars. The 'ModelData' script must be called to read the simulation data and apply it so that the avatars express emotions or carry out actions. Furthermore, the scene must also be created. This scene contains the environment the avatars are in, which can be tailored to the specific model. An argument between two people can occur in a house, but a model describing a different situation might appear somewhere else. Finally, for the virtualisation to be viewed, a few cameras must be set up and controlled to show the relevant avatar. The approach contains some templates for the virtualisation script, so it is easier to configure.

5 The Designed Virtualisation for the Example Network Model

Inside the Unity project, a new scene was created. The first part that must be created is the environment in which the avatars are placed. This process can be extensive however is skipped because it is not necessary for testing the libraries. Therefore a simple cube is created, which acts as the ground. Next, two avatars are placed into the scene facing each other. A few lights and cameras are added to make the avatars visible. Finally, an empty game object is added to the scene. Later, a script will be added to this invisible object which will automatically run when the scene is started.

The next step is to create the virtualisation script. Here the simulation data is read using the 'ModelData' script, and the avatars are controlled with the read data. This script is attached to the empty game object created in the scene. In this scenario, there

are two avatars, and a reference to both avatars is added. Person A will be avatar A, and person B will be avatar B from this point on.

The actions of each avatar need to be implemented since these can be specific to the model. In this example, a couple of activities are performed by avatar B. Namely, they look away and walk away. Therefore, two actions need to be created, a gaze-away action and a walk away action. The latter can be added to the 'Actions' library utilising a new class. This new class implements the abstract 'AvatarAction' class making it easy to call later. The implementation of this action has access to avatars components which enables anything to be modified. The avatar will need to turn around and move forward, implemented by rotating the Transform object and applying force to the Rigidbody of the avatar. Additionally, a few animations can be triggered to make the action look more realistic.

The second action is the avatar B's gaze-away. It was implemented in the same way as the walk away action. A new class was added to the 'Actions' library in which the eyes move to the right and down, utilising the eye movement library. Along with this, the avatar also rotates a couple of degrees to the side.

With all the behaviours implemented, the script should create a step function where each row of the simulation data is read and applied to the corresponding actions of the avatars. In this example, three values in a row are applied. Avatar A's anger, a value between 0 and 1 generated from the causal model, represents the intensity of the anger emotion. This value is virtualised by accessing avatar A's emotion controller and setting the 'anger' emotion to the intensity. Next, avatar B's walk away, also a value between 0 and 1, is applied. Once this value is larger than 0.5, the action is triggered, and the avatar walks away. A simple check is added in the step function, and once the condition becomes true, the action is initiated. The gaze-away value is very similar to the anger value. In each step, the gaze-away action is set to the value representing the intensity. A gaze-away value of 0 means the avatar is not looking away at all, while a value of 1 means they are entirely looking away.

The step function is triggered in the virtualisation script once and then waits for 200ms and steps again. The step function is repeated until all the simulation data has been applied. Figure 4 shows how the libraries are used in the virtualisation of this example. Avatar B has two additional actions that were added specifically for this example. The virtualisation script also uses a camera controller script that sets the active camera to the avatar currently performing an action. When avatar B walks away, the camera is positioned, so both avatar A and B are in frame. However, when avatar B is looking away, the camera showing only that avatar is triggered.

Some subsequent snapshots of the video displaying this virtualisation of the scenario for the causal mental network model discussed in Sect. 3 are shown in Fig. 5. An mp4 video displaying this virtualisation case can be found as Linked Data at URL https://www.researchgate.net/publication/354200343.

6 Discussion

As part of the CoSiHuman project (Treur et al. 2021), the work reported in this paper aimed to explore creating virtual avatars using the Makehuman software, Blender, and

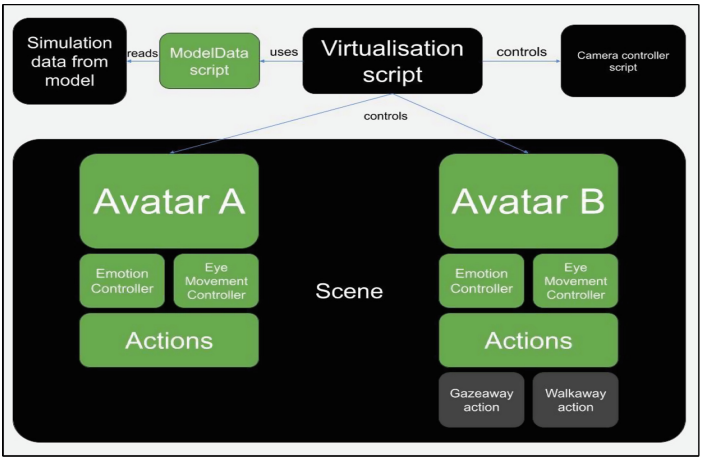

Fig. 4. Diagram of the example virtualisation using the avatars and libraries.

Fig. 5. Snapshots from the virtualisation of the scenario from Sect. 3.

Unity software and to make these avatars express human-like traits using causal models. Furthermore, a goal was to create avatars and libraries that can be used in several different virtualisation examples. The outcome shows that making these avatars is possible, and applying them to simulation data works. The avatars are also dynamic, as they can act out several behaviours in many different scenarios. Since the software used allows

a lot of customisation, creating new and different looking avatars is straightforward, albeit additional tools must be developed to make this less time-consuming. Building the approach further is straightforward as lots of documentation is provided, but the two avatars already created can also be applied to different models. One of the goals was to make it easy for the approach to be expanded, and laying out the creation process should allow for easy adaptations. Since many different software components were used for avatar virtualisation, any part can be modified and changed. For example, say an avatar requires very precise animation of the hand gestures with animations; an avatar can be created using the Makehuman software with a more detailed skeleton. At the same time, all blend shapes can stay the same. Inside Blender, more animations can be added as required. Another example would be facial expressions. If more emotions are needed, or existing emotions need to be modified, the changes can be performed in Makehuman while the remaining export process remains the same.

However, creating avatars is not yet as easy because it is very time-consuming. After the initial avatar is created in Makehuman, the facial expressions must be applied and exported. This process is very repetitive and provides an ideal opportunity for automation. A new part of the software can create a base avatar created in Makehuman and automatically applies the blend shapes and animations. While this process might take a bit of time, it should not require any more input from a human. This automation dramatically improves the flexibility of this virtualisation approach, as the creation of the avatar is very easy with an already existing graphic user interface with lots of customisation.

A vital feature of this approach is to make the avatars relatable to the humans observing them. The avatars must elicit empathy, which can be achieved in several different ways—for example, matching the appearance to the individuals interacting with the avatar. The most straightforward manner to accomplish this would be to match the gender. Also, making the avatar more human-like is effective. In the work reported in the current paper, a focus was on exploring the possibility of virtualisation. Fortunately, the Unity game engine allows for a vast amount of graphical features. Making the environment feel more authentic will be essential and is possible. Furthermore, making the avatars behave more human-like is something that can be improved. More natural features can be added, like blinking at random intervals or turning the head slightly every once in a while to make avatars less robotic. Avatars can also change poses sometimes.

Additionally, several libraries were created to make controlling the virtual avatars easier. These include the emotion controller, eye movement controller, and the avatar actions library. Each component is responsible for making it easy to set the properties of the avatar. Setting emotions, moving the eyes, and triggering actions are easy to use by abstracting the complex implementation. These libraries can also be built on and expanded with relative ease.

Overall, the aim of this approach to create virtual avatars was accomplished. The avatars were also applied to simulation data from a causal model showing that it can work. This was further supported by the two experienced software developers from industry who evaluated the approach. They provided very positive feedback and described the approach as well-documented and ready to be expanded in the future. Furthermore, several other further prospects came into view: Automating avatar creation, additional realism features and making the avatars more human-like. Along with this, allowing

bidirectional interactions between humans and avatars also becomes a possibility. In therapy, as a role-playing tool, humans can interact with the avatars not just by observing but by making choices on the fly. This interaction can be in choosing a different emotion regulation method, which in turn causes the avatar to behave differently. Selecting a different emotion regulation method is particularly useful to show how other behaviour changes the avatar's actions. Furthermore, the possibilities for expanding this approach are virtually endless, and applications to the real world are extensive.

References

Blender Foundation. Blender - a 3D modelling and rendering package. Stichting Blender Foundation. Blender Foundation, Amsterdam (2018)

Blender Foundation. Introduction — Blender Manual, Docs.blender.org (2021). https://docs.blender.org/manual/en/2.90/animation/shape_keys/introduction.html, Accessed: 12 July 2021

Burgos-Robles, A., Gothard, K., Monfils, M., Morozov, A., Vicentic, A.: Conserved features of anterior cingulate networks support observational learning across species. Neurosci. Biobehav. Rev. **107**, 215–228 (2019). https://www.sciencedirect.com/science/article/pii/S0149763419301393

Ekman, P., Cordaro, D.: What is meant by calling emotions basic. Emot. Rev. **3**(4), 364–370 (2011). https://doi.org/10.1177/1754073911410740

Ekman, P., Friesen, W.: Unmasking the Face. Malor Books, Cambridge (2003)

Ekman, P., Hager, J., Friesen, W.: Facial Action Coding System. Research Nexus, Salt Lake City (2002)

Essau, C., LeBlanc, S.S., Ollendick, T.H. (eds.) Emotional regulation and psychopathology in children and adolescents. Oxford University Press (2017)

Gilbert, M., Demarchi, S., Urdapilleta, I.: FACSHuman, a software program for creating experimental material by modeling 3D facial expressions. Behav. Res. Methods **53**(5), 2252–2272 (2021). https://doi.org/10.3758/s13428-021-01559-9

Hall, L., Woods, S.: Empathic interaction with synthetic characters: the importance of similarity. Encyclopedia Hum. Comput. Interact. (2005). Idea Group Publishing

Kim, J.: Philosophy of Mind, 3rd edn., pp. 104–123. Westview Press, Colorado (1996)

Makehumancommunity.org. Documentation: What is MakeHuman? - MakeHuman Community Wiki. (2016). http://www.makehumancommunity.org/wiki/Documentation:What_is_MakeHuman%3F, Accessed 26 Jul 2021

Miller, H., Jr.: The SAGE Encyclopedia of Theory in Psychology, pp. 249–250. SAGE Publications, Inc., Thousand Oaks (2016)

MCV. Digging into rigging: Mixamo's character service explored | MCV/DEVELOP, MCV/DEVELOP (2021). https://www.mcvuk.com/development-news/digging-into-rigging-mixamos-character-service-explored/, Accessed 12 Jul 2021

Preston, S., de Waal, F.: Empathy: its ultimate and proximate bases. Behav. Brain Sci. **25**(1), 1–20 (2002)

Singer, T., Tusche, A.: Neuroeconomics (Second Edition), 2nd edn., pp. 513–532. Academic Press, San Diego (2014)

Treur, J.: Network-Oriented Modeling for Adaptive Networks: Designing Higher-Order Adaptive Biological, Mental and Social Network Models. Springer Nature Publishing, Cham (2020)

Treur, R.M., Treur, J., Koole, S.L.: From Natural humans to artificial humans and back: an integrative neuroscience-AI perspective on confluence. In: First International Conference on Being One and Many (2021). https://www.researchgate.net/publication/349297552