D6.6 Motion Stylization Implementation

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	December 31st 2019 [M24]
Actual delivery date	December 20th 2019
Leader	UPF
Reply to	josep.blat@upf.edu
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	Smart Asset re-Use in Creative Environments
Document name	D6.6 Motion Stylization Implementation
Security (distribution level)	PU
Contractual date of delivery	December 31st 2019 [M24]
Actual date of delivery	December 20th 2019
Deliverable name	Motion Stylization Implementation
Type	DEM
Status & version	Submission Version
Number of pages	21
WP / Task responsible	UPF
Other contributors	-
Author(s)	Josep Blat, David Moreno, Javi Agenjo, Hermann Plass
EC Project Officer	Ms. Adelina-Cornelia DINU, Adelina-Cornelia.DINU@ec.europa.eu
Abstract	This report accompanies the public demonstrator of Motion Stylization Implementation carried out by the UPF-GTI, presenting the overall methodology and approach and details on the implementation towards achieving virtual characters identity through motion stylization. A guide to the use of the demonstrator, a standalone tool called *Medusa* is included as an annex to this report.
Keywords	Motion stylization, Virtual Character Identity, Behaviour
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
14th November 2019	Initial version with substantial content, main contributor, David Moreno.
1st December 2019	Improved version with revisions and enhancements by David Moreno, and contributions from Herman Plass and Javi Agenjo.
6th December 2019	Revised version for internal review by FA
13th December 2019	Post-review by FA addressing their comments
20th December 2019	Final version for submission

Table of Contents

EXECUTIVE SUMMARY **5**

BACKGROUND **5**

INTRODUCTION **5**

Main objectives 5

Context and solutions 6

Methodology 6

DESCRIPTION OF THE TOOL AND ITS IMPLEMENTATION **7**

Web based tool for background characters 7

Behaviour 7

Animation Controller 9

Open source library for behaviour creation 10

Open repository of behaviours 11

WebGLStudio development and integration **11**

Discussion, conclusions and further work **12**

Self Assessment 12

Conclusions and further work 13

References **13**

Trademarks and Copyrights **14**

Acronyms and abbreviations **14**

Annex: Medusa user's guide **15**

1 EXECUTIVE SUMMARY

This report accompanies the public demonstrator of the *Motion Stylization Implementation* carried out by UPF-GTI within WP6T5 of the SAUCE project. The demonstrator is available at https://webglstudio.org/users/dmoreno/projects/saucemedusa/. The report discusses the methodology around motion stylization adopted by the UPF-GTI, mainly consisting of web-based tools and pipelines, and adopting a separation of concerns expressed through two different modules:

- Motion stylization through behaviour: what to do in a given scenario
- Motion stylization through animation tweaking: how to perform the animations and interactions

Details for the implementation are provided in different sections as well as for the tasks related to creation of smart interest points enable that characters behaviours are environment aware. A repository for behaviours which can be shared is another target, while the libraries are also shared via GitHub. This intends to support the openness of the tools within easy to use pipelines, and able to be integrated in different alternatives, such as the Virtual Production tools of partner Filmakademie.

The tools and pipelines support the key purpose of the task, which is to cater for the generation of uniquely distinct virtual characters for Virtual Production or for game industry.

The guide to use the standalone tool produced, *Medusa*, is an annex of the report.

2 BACKGROUND

The SAUCE project overall goal is to facilitate the re-use of assets and to maximize the potential for re-purposing content for the creative industries. One of the key concepts to achieve the goal is smart asset, and making existing assets smarter by labeling and/or transformation is part of the contribution. This is especially interesting for animated content, e.g. character animation, which is a time consuming and labor intensive task. Specifically Virtual Production scenarios appear to be ideal candidates for such novel concepts, where animations need to be available in real-time. Workpackage 6 *Semantic Animation Production* combines the effort of semantically driven character animation and Virtual Production.

This deliverable is provided within the context of WP6T5, *Motion stylization*. The deliverable discusses web-based tools developed by UPF-GTI which can be used for motion stylization of background characters, within a pipeline which can facilitate annotation and editing, and collaborative workflows, making animations smarter and re-usable, as it will be detailed in the following sections. While the tools can be standalone, they are also integrated in the interactive 3D graphics and environment *WebGLStudio* [AEB13] created by UPF-GTI (https://webglstudio.org/, and https://github.com/jagenjo/webglstudio.js in GitHub).

The link to the demonstrator is: https://webglstudio.org/users/dmoreno/projects/saucemedusa/.

3 INTRODUCTION

Within WP6T5, UPF-GTI has focused their efforts on working on the development of pipelines and tools to generate smart assets. More specifically, web based pipelines and tools to generate character's identity through motion stylization and context awareness. We mean by characters' identity the decision virtual characters take, and how they perform the actions according to local or global properties. In the big topic of motion stylization, we have been working on two different branches, namely, motion stylization through the behaviour of the virtual entities and their animation stylization.

3.1 Main objectives

In summary, the overall main objectives of this deliverable are:

- To provide accessible pipelines and tools to generate virtual characters identity under dynamic scene constraints in real time.
- To provide models of character stylization

3.2 Context and solution

The last two decades have seen a growing trend stressing the importance of creating realistic virtual agents in video games, animated productions or digital films, and this importance is expected to increase much more in the near future. Obtaining realistic virtual agents is not only a matter of appearance, this is just one of several components within other different human aspects that make a virtual agent realistic; another key one is the behaviour it could have in different scenarios. Another important aspect to consider is the possibility to recycle and reuse virtual content aiming to ease the development of new assets and reduce the large amount of almost identical data.

Several tools and investigations on finding the most suitable methodology to generate personalized behaviours are in progress or already available. The existing solutions try to achieve a balance between quality and accessibility/ease of use. *Unity*[1] and *Unreal Engine*[2] provide robust solutions to these problems. But their solutions are proprietary, so that the collaborative development is not possible. There exist open source solutions, such as Owl-bt[3], which is an editor of Behaviour Trees that claims to be inspired by Unreal Engine behavior trees, but it does not include 3D simulation nor an animation system, and thus it is of limited usefulness. The solution presented in this report, *Medusa*, aims at incorporating the advantages of these already existing and proven tools, but adding the features of being an open web based tool, connected to an open repository, so that it can make the content and tools more accessible and collaborative, as wall as facilitating its improvement by anybody who wants to.

3.3 Methodology

In general terms, the option of interactive web-based tools is grounded on the possibility that WebGL provides to deliver good quality interactive 3D graphics on the web [ERBAB14]. The advantages are significant, and a key one is the perspective of democratization afforded.

A second aspect of the approach is to develop the tools through several cooperative modules, where each covers a different requirement of those identified; this ensures the scalability of the approach, as the tools can be expanded. Trying to deal with the whole topic without dividing it in different pieces would have led us to too complex problems, and consequently, to complex solutions, which would have been completely inconsistent with one of our main goals of our development, which is to deliver easy-to-use pipelines and tools.

Within each module we adopted a specific strategy related to optimising the goals of its implementation, obtaining the most flexible solutions, and prepare for any future improvements in an unconstrained way. In the next section we explain in detail the methodology used for the internal approach to each module, as the motion stylization through the behaviour or through the character identity (physical properties, emotional state, etc.).

As previously mentioned we have been tackling two different concepts: the motion stylization through the behaviour of the virtual entities, and through their representations within a 3D scenario. The reason of dividing the motion stylization concept into these two concepts is that decoupling the topic permits to deepen each specific aspect of the character identity. Additionally, decoupling the topic allows us to create more specialized tools for each concept of motion stylization, and then being able to combine both and achieve more successful results.

The implementation of the solution for the motion stylization through the behaviour of the virtual agents is based on the development of a behaviour tree based engine and editor. The reason for

[1] https://unity.com/. Unity is a cross-platform game engine developed by Unity Technologies.
[2] https://www.unrealengine.com. UE is a game engine developed by Epic Games.
[3] https://www.npmjs.com/package/owl-bt

using the Behaviour Trees [S19], [FGGDG09] as a basis is that it offers much more freedom and transparency than other technologies, such as UtilityAI [G19] or Deep Learning approaches, which have other really interesting properties for controlling NPCs (secondary characters), but not as interesting as Behaviour Trees for widening the range of users. Furthermore, other technologies or algorithms can be embedded in graph nodes, so there is the possibility of taking advantage of novel real time artificial intelligence methodologies adopting them directly from the behaviour tree.

4 DESCRIPTION OF THE TOOL AND ITS IMPLEMENTATION

4.1 Web based tool for background characters

UPF-GTI has been developing easy-to-use web based tools and pipelines intended to animate, in real time, multiple secondary (or background) 3D virtual agents, where quantity is more important than quality in terms of the results. This activity would complement the focus of IK related to very high quality animation of foreground or main characters. We aim at giving them an intelligence that makes them aware of their surroundings and can navigate in a scenario semi-autonomously. Further, we intend that every entity achieves behaviors that differ from the rest of the entities even though the action is the same, giving some personality, taking into account character properties, and finally having a realistic background scene, populated with each person behaving uniquely. This is achieved by the stylization of the actions of the virtual characters.
As mentioned in the *Introduction* section, we have been developing two different approaches to motion stylization; the behaviour of a specific agent in a specific environment and context, and the style of the character when performing an action, that we discuss next in different sub-subsections.

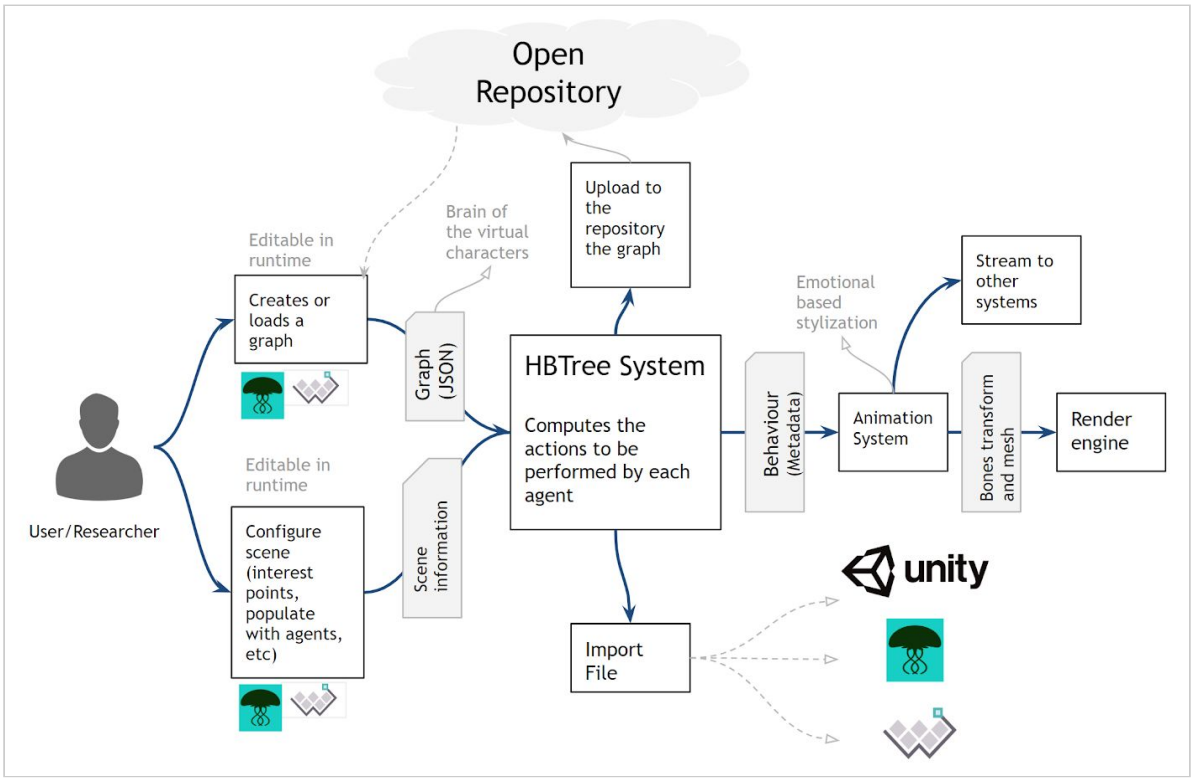
4.1.1 Behaviour

Referring to the virtual characters' behaviour, our efforts have been focused on providing a web based pipeline and tool which allows to construct, edit and reuse behaviours for multiple virtual agents. Within the key goal of creating intuitive interfaces and workflows, all the tools developed within WP6T5 are graph based. Thus, the tool to generate the behaviours of the virtual agents includes a graph based editor, so the process of creating the AI of the characters is much simpler and understandable, but equally effective. The main functionalities of the tool in its current state will be explained next.
The standalone tool, *Medusa* in further mentions, allows the user to visualize in real time the representation of a significant number of virtual characters in a schematic 3D scenario, interacting with some smart elements in the scene. We refer to smart elements as some interest points in the environment which could have some kind of intelligence, properties or information to give to those virtual characters when they reach the place of this point of interest. In that way, it is possible to create multiple and varied use cases. To do so, there are three main modules; the render engine (based on WebGL), the animation engine and the behaviour module. This last one executes the graph created in the graphical editor. Obviously, as a web tool, it has other important modules which make the tool more robust and complete, as importing and exporting systems, or modules which provide intuitive interaction with the 3D world.
The basis of our development for controlling the virtual agents are **Behaviour Trees,** but we have introduced some modifications which enhance their possibilities. We have named the modified Behaviour Trees **Hybrid Behaviour Trees** (HBT in further mentions): our implementation combines the workflow of a flow diagram and that of Behaviour Trees, as well as some specific nodes developed for virtual characters. We have chosen as basis the Behaviour Trees due to the fact that are understandable, easily scalable and new techniques or technologies can be included, embedded inside their nodes. For example, an Utility algorithm could be integrated inside a node, which in fact is

part of our future plan of work. The current status of the HBT system includes a repository of nodes, which could be classified in five categories:

- **Composite** nodes: To determine the execution of their children
- **Decorator** nodes: To determine whether a branch has to be executed or not
- **Action** nodes: Compute (but not execute) the acton a virtual character has to perform
- **Hybrid** nodes: Computed in the flow phase of execution (before the execution of the whole HBT) to dynamically change properties of the behaviour tree nodes in real time
- **HBTProperty** nodes: Used by conditionals or used to compute some operations and use the results to make some verifications. In that way, the property can be used not only to read their value, but to use it as the artist wishes.

In the next image, there is a simple representation of the workflow of the system and tool usage.



Medusa workflow diagram

To make this tool more valuable and useful it should be open towards further developments. Thus, we have developed importing/exporting functionalities, so that any scenario configuration or any behaviour can be exported or saved in an open repository in a way that everybody can have access to it. The whole HBT is saved in a JSON file, containing all the relevant information of each node of the graph to be easily set up in future uses. Therefore, when some artist or developer is looking, for instance, for a rainy street behaviour for some virtual characters, the behaviour already developed by someone else (or him/herself), and s/he will only have to modify what is required in this case, avoiding to develop from scratch the whole behaviour.

The usefulness of exporting the behaviour is not limited to this web tool, as a porting of the system to C# and C++ is under development, so it can be used in external tools or in independent developments, having the possibility of doing it in JavaScript, C++ and C#.

In Medusa it is possible to configure the 3D scenario, as mentioned previously. In the schematic scenario created inside the tool, the virtual characters have to exist and perform whatever is

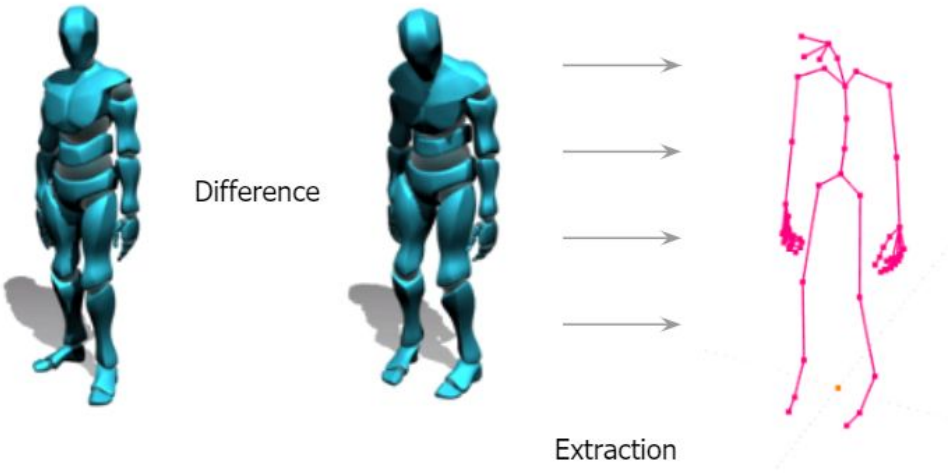
determined by the behaviour control module. Thus, it is possible to create smart interest points, which will give special properties to whoever reaches that place in the simulation, and also it is possible to create paths for those virtual characters which do not have any special task to perform and they just have to walk around the scene.

4.1.2 Animation Controller

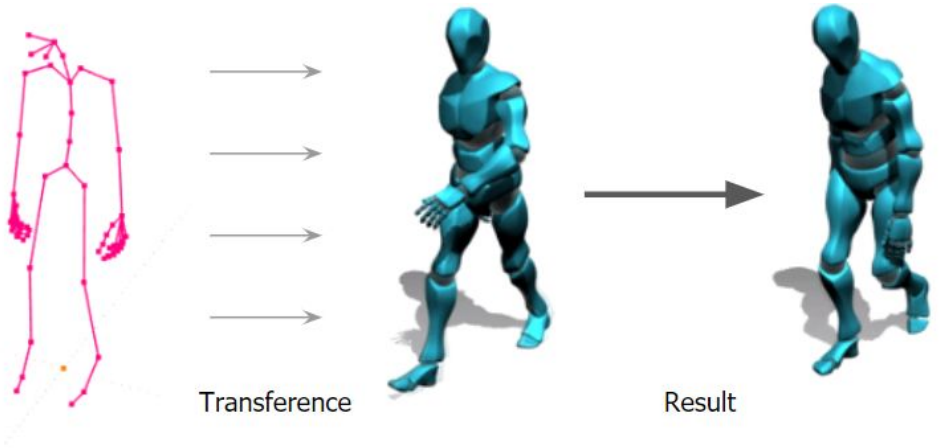
This is another layer we have been working on. It is the layer below the Hybrid Behaviour Tree. Once the HBT has computed what a virtual character should do, there is a need for an interpreter and executor of this result. The animation controller, divided into several sub-layers, is the module in charge of interpreting the result of the HBT execution and, after that, its execution. For example, a HBT outputs "move to a certain position urgently". Then, the animation controller will interpret that output as several steps: set the target position to the value returned, and also put the urgency property to true or to a high value. Afterwards, this will be sent to the Animation State Machine (on which we are currently working and which will be explained in further sections) to decide the animation to apply. Finally, it will be sent to the animation stylization module, where the emotions and personal properties are taken into account to stylize the motion of the virtual character by tweaking the animation and the body pose.

4.1.2.1 Animation Stylization

The final layer of the animation controller, and one of the most relevant for this project, is the stylization of the action realized by the virtual character. This means that the same action can be performed in a number of different ways, depending on several character parameters or properties.
Specifically, UPF-GTI has developed stylizing the animations according to the emotion of the virtual character which is being animated. We have added to WebGLStudio features which allow to extract the style (or emotion) of a specific animation, whose style is also previously known, and in the next step transfer this style to a different animation. An example of this could be extracting the sadness factor of an *Idle* animation, and transferring that sadness to a *Walking* animation.



Extracting the difference between a neutral idle and sad idle



Transferring the difference between a neutral idle and sad idle to a neutral walking

This extraction-transference is performed by extracting the proportional rotation difference in the relevant bones of a skeleton between a neutral animation and a styled one (e.g. neutral Idle and sad Idle). In our development, we consider as relevant bones the whole spine, the shoulders, and the head, parts of the human body which can express a wide range of emotions only through the pose.
Furthermore, abstracting the stylization and being able to stylize animations without having to extract styles from previous ones is desirable. That is why a first model (or mapping) of tweaks has been set up, associating emotions to a particular set of modifications on the virtual character body pose. This mapping has been set according to some emotion recognition from human body pose researches [DMS12], and from our own experience. In the next phase we envisage to change this mapping for an approximation to the Emotion Wheel, where more emotions can be controlled at the same time through Nearest Neighbour Interpolation. This will be based on identifying more precisely the movement itself, the strength used by each bone of the virtual character to perform each action.
Other techniques which are currently being developed will help to cover a wider range of use cases. The first one would be the retargeting of skeletons or animations. By implementing a retargeting, we will be able to apply actions from different virtual characters with different number of bones or different rigs, achieving all the advantages it involves. Another implementation that we are currently carrying out is the masking of animations. The masking is the division of the body in different parts, to be able to treat them independently, so that we can apply different actions in different parts of the body, getting more combinations. For example, we can animate the hand with a gesture, while the lower body is walking, and the other hand is holding an object.

4.2 Open source library for behaviour creation

The Medusa development is open source, and the code is available on GitHub. Furthermore, the behaviour of the virtual characters is determined by the HBTree module, which is an independent Javascript library. This means that besides working inside the standalone tool, it has been designed to be embeddable inside other developments as well, just by calling the main creation and updating methods.
The structure of the code has been designed to be as clear and clean as possible, and the methodology to create new nodes with new functionalities as understandable as possible as well, so that external developers are able to grow and improve the library along the time and usage, making it more versatile and complete with new nodes and algorithms. It is based on the LiteGraph.js library [Lite], developed by Javi Agenjo (UPF), and used in the WebGLStudio engine. This JavaScript library is also available on GitHub, and it is regularly updated and improved. Litegraph is a library in Javascript to create graphs in the browser similar to Unreal Engine Blueprints, with lots of useful features for several fields of 3D graphics.

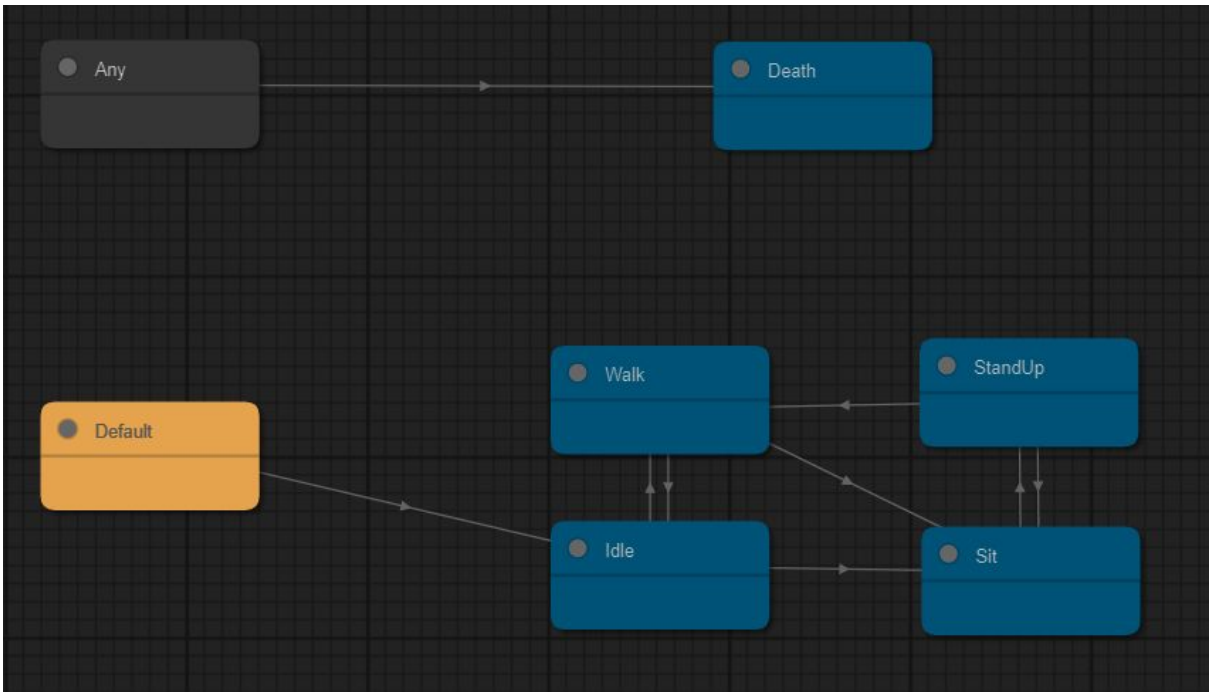
Within the ongoing porting of these libraries to C++ and C# , the HBTree library could be used in the most widespread engines, such as Unity or Unreal Engine, so that one can benefit both from the ease of use of Medusa with the power and render quality of those engines.
To allow independent developers to fully integrate the library to their system, UPF-GTI has developed a facade (which has the functionalities of an API, but permits the overwriting of some of the methods), where the new developer should implement the methods in order to have the full functionality of the HBTree system. This is due to the fact that many operations inside some nodes have to be aware of the context or some agent properties, and each development will have their own methods to access to the environment information or to the agents.

4.3 Open repository of behaviours

There is currently a scarcity of open and readily available virtual characters behaviours, and one of our goals is to increase the amount of data of this type available. Thus, we have been developing a proof of concept of an open repository of behaviours, where ourselves, and potentially artists and developers could upload their work so other people in further developments can use it directly or use it as a starting point for their developments.
The behaviours will be tagged when uploading them (manually, as tagging a graph automatically is a challenging task), as well as validating them, so that the artists or users can find what they are looking for more easily.

5 WebGLStudio development and integration

While *Medusa* is a standalone tool, where tests are carried out and use case support is tested, the tool is based on the core technology of the open platform developed by UPF-GTI over a number of projects, which is intended to facilitate the use of advanced interactive 3D graphics on the web, with less expertise required.
To meet the SAUCE project requirements, several improvements of the core animation system of WebGLStudio have been carried out. A new file format has been created for animation resources, with the extension .skanim (from skeletal animation), which leads to a better performance and a faster read of the animation keyframing. As a consequence of this format, a novel animation system (specifically implemented for skeletal characters) has been developed and integrated in the platform. A file format for skeletal virtual characters has been created, suitable for a later conversion to a binary format, which allows to load virtual agents way faster than the previous format.
UPF-GTI plans to integrate the HBTree.js library within the WebGLStudio platform, to provide very flexible and powerful behaviour control of animation. This will be carried out, according to a plan already designed, when it reaches a more stable implementation, as it has been continuously experiencing updates to make it more robust and complete. The same will happen with the HBTree Editor and the Human Body Pose Stylization modules.
Decoupling the behavior planning from the realization of that behavior has been going on in the animation controller: the behavior trees decides how this agent is going to behave from a given state producing a new state; and later, this state is read from different components of the engine, in this case the animation controller, and reproduces the animation that fits best depending on that state. For this purpose, UPF-GTI planned and developed a modular *state machine* library and an editor integrated in WebGLStudio. It comes with a basic set of nodes that can be extended through the currently available scripting component, as shown and illustrated in the following figure and diagram.



Capture of the WebGLStudio State Machine editor

The integration of this state machine editor is split into three different parts: the library, the webglstudio component and lastly the FSM editor for webglstudio (UI).
Independently to WebGLStudio we have implemented a FSM library which can be included into any project. This library has three main classes, the FSM core, transitions and nodes.

The **FSM Core** comprehends a set of attributes (the state), a list of nodes and transitions between nodes depending on the state. All this states, nodes and transitions are serialized into a *.fsm* file so it can be stored and shared.

Each **node** may have specific behaviours that could include running some component or modifying the state. This set of behaviors can be extended registering new types of nodes through the *registerNodeType* method. Each node could implement 3 actions: *onEnter*, *onUpdate* and *onExit*. *onEnter* and *onExit* actions are triggered when transitioning between nodes.

Nodes can be interconnected each other using **transitions**, the currently evaluated node checks if **any** transition returns true in order to change to the new node. Each transition has 1 or more conditions, only if all conditions in this specific transition evaluates to true the transition check is passed. The condition types could be number, booleans, vec2, vec3, vec4 or registered trigger actions.

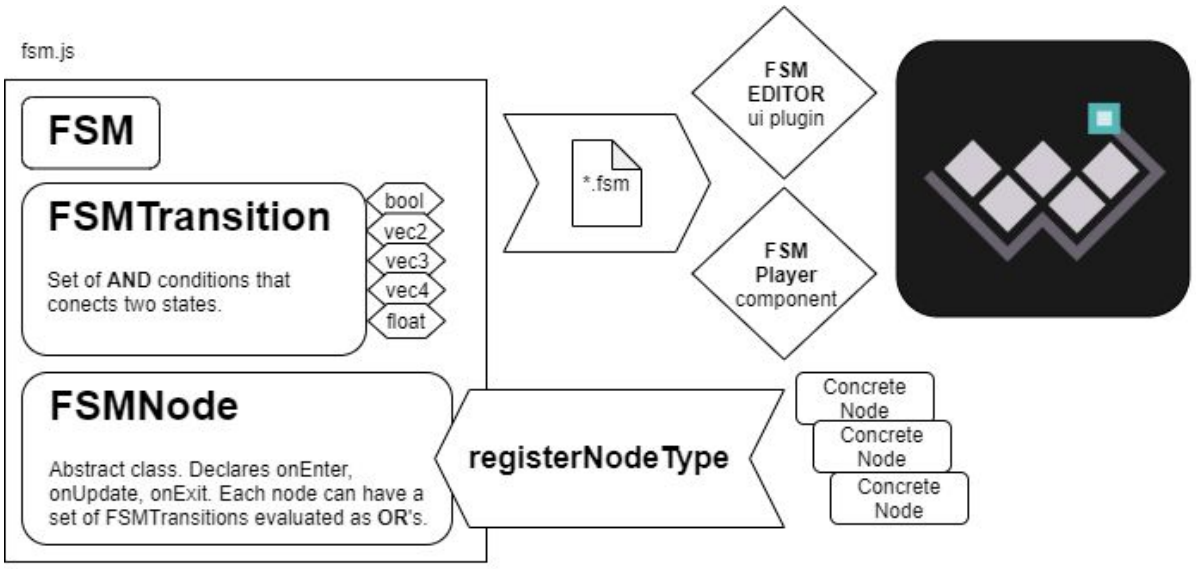


Diagram of the architecture of the Finite State Machine and its components

6 Discussion, conclusions and further work

6.1 Self Assessment

As defined in the Description of the Action, the goal of the task 5 within WP6 is to create motion stylization tools on the WebGLStudio editor and add annotation tools and collaborative editing to support novel collaborative workflows. Once the features of each given style are defined, they can be transferred to other animations and used to create variations of existing animations.

Success indicators:
As *advance on the state of the art*, automatic provision of animation variants from neutral animations has been developed in the WebGLStudio editor as an independent component, where style from one animation can be transferred to another one. A modular approach based on an evolution of common Behaviour Trees to control the actions, provision of context aware virtual characters, as well as an an animation controller with a stylization module to carry out the behaviours, indicate the success of the task envisaged.
As for *technology improvement*, the development of a first version of such tools in the (real-time) web environment, called Medusa, shows that the target TRL increase from an initial stage at TRL 1-2 to lab validated tools (TRL 5) has been achieved. Further validation will take place within the field trials and user evaluations in experimental productions of WP8, and consequent enhancements of the tool, as well as a stronger level of integration within WebGLStudio are expected.
Tests to verify the usefulness of our implementations are in progress, such as a proof of concept of a video game which covers the majority of use cases that we aim to cover. Also we shall intend to test our implementations in a Video Game Creation Master, through students considering using our tool.

6.2 Conclusions and further work

Motion stylization of virtual characters can be achieved in several ways. Our approach tries to separate two basic concerns, so that they can can be individually edited, and linked again later. The overall goal is to avoid as much identical behaviours as possible for the different characters, and intends to create unique identities.
The current version of the tool is is not completely robust. There are some features that must be polished, and some other ones might be added. Similarly, the libraries are growing every day, to cover more use cases and to be embedded more easily in external implementations. Furthermore,

formalizing and extending the open repository of tagged behaviours is targeted with the goal of making it grow as soon as possible, in the context of creating a community of users as well which will use this behaviour for different purposes (game development, realistic street simulations, etc).
The next key steps along the implementation of the tool are to find a suitable mapping which includes more emotions more precisely, and adapt this mapping to the motion of the actions, not only on the body pose along all the performance. This means that not only the body pose is affected, but the strength/energy of the movements are affected too.
These perspectives will be included in the next steps of the SAUCE project, where integrability, usability and fitness for purpose in the context of professional use of UPF-GTI tools together with other tools generated will be tested. In particular, integration with FA virtual production editing tools (VPET) and the use of Medusa in the context of the production of a simulated game will be carried out, as discussed in deliverables D8.2.

7 References

[AEB13] Javi Agenjo, Alun Evans, and Josep Blat. 2013. WebGLStudio: a pipeline for WebGL scene creation. In *Proceedings of the 18th International Conference on 3D Web Technology (Web3D '13)*. ACM, New York, NY, USA, 79-82. DOI: https://doi.org/10.1145/2466533.2466551
[DMS12] Dael, N., Mortillaro, M. & Scherer, K. R. 2012. Emotion expression in body action and posture. *Emotion* **12**(5), 1085-1101.
[ERBAB14] Alun Evans, Marco Romeo, Arash Bahrehmand, Javi Agenjo, and Josep Blat. 2014. 3d graphics on the web: A survey. *Computers & Graphics* **41**, 43 – 61.
[Lite] https://github.com/jagenjo/litegraph.js
[S19] Simpson, C. Gamasutra: Chris Simpson's Blog - Behavior trees for AI: How they work. https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php. (Accessed on 06/04/2019).
[FGGDG09] Flórez-Puga, G., Gomez-Martin, M. A., Gomez-Martin, P. P., Díaz-Agudo, B.& González-Calero, P. A. 2009. Query-enabled behavior trees. *IEEE Transactions on Computational Intelligence and AI in Games* **1**, 298–308.
[G19] Graham, D. R. GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf. http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf. (Accessed on 06/04/2019).

8 Trademarks and Copyrights

Unity and Unreal Engine are trademarks used within this report.

9 Acronyms and abbreviations

AI - Artificial Intelligence
HBT - Hybrid Behaviour Trees, evolution of current Behaviour Trees
FSM - Finite State Machine

Annex: Medusa Guide

General Idea of the tool **16**

Medusa guide **16**

Mode buttons 16

Navigation Bar 17

Graph Inspector 17

Graph based Editor Area 18

Right Inspector 19

Hybrid Behaviour Tree Nodes 19

Common Behavior Tree Nodes 19

Conditional Node/BoolConditional Node 19

Sequencer Node 20

Selector Node 20

LineOfSight Node 20

SimpleAnimate 20

LookAt Node 20

SetProperty Node 20

Wait Node 20

MoveTo Node: 20

Flow Nodes for the Behavior Tree 21

EQSNearestInterestPoint 21

EQSDistanceTo 21

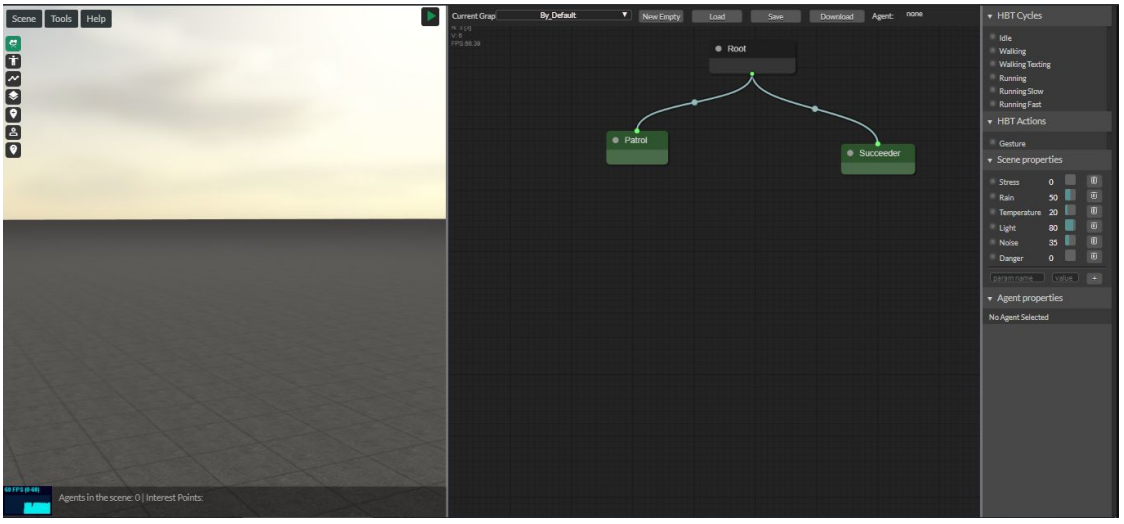
EQSNearestAgent 21

General Idea of the tool

UPF-GTI has been developing easy-to-use web based tools and pipelines intended to animate multiple secondary (or background) 3D virtual agents (where quantity is more important than quality) in real time, by giving them an intelligence that makes them aware of their surroundings, so that the agents can semi-autonomously navigate in a scenario. Further, we work to achieve that each one has behaviors that differ from the other entities even though the action is the same, giving it some personality, taking into account character properties. Finally resulting in a realistic background scene populated with each character behaving uniquely. This is achieved by the stylization of the actions of the virtual characters. Medusa is a web-based standalone tool reflecting these goals. The tools are available on GitHub (https://github.com/upf-gti/Sauce).

Medusa guide

The overall interface of Medusa is shown next, followed by details on its different sections.



Mode buttons

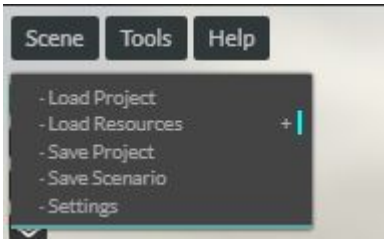
The "mode" buttons allow the user to change the interaction with the 3D environment.

- The default is the *navigation mode*, moving the camera though the scene and being able to click on agents and interest points to get information. Also if an Agent is selected, it can be removed from the scene by pressing the "back" key .
- When the *Agent creation mode* is selected, the user can add virtual characters to the scene just by clicking on the position one wants the character to be.
- The third button enables the *path creation mode*. By clicking on the floor, control points will be added. When finished, the user should click the end path button that will appear in a floating dialog. Virtual characters currently follow the nearest path they find. In future updates, it will be possible to select which path each agent has to follow
- The fourth button is the *smart area creation mode*, currently in development.
- The *Interest Point creation mode* is next. When it is enabled, Interest Points can be added to the scene by left click on the position we want to add it. Afterwards, a floating dialog will appear to configure such an Interest Point, the properties it will affect, the name

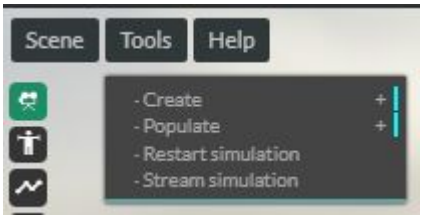
and the type.

- The two last buttons are used to visualize information on the 3D canvas. Specifically, to see labels identifying the virtual characters and all the interest points

Navigation Bar



The first navigation bar option is *Scene*. In this section we can find the functionalities to load a whole project (including agents, paths, interest points, behaviours), load resources individually (scene setup, animations), save the whole project or save just the scenario configuration and finally check and change some settings. Currently, there are no relevant changes in the settings section, just preferences of visualizing the scene.



The following option is *Tools*, which contains some of the important features of the tool, apart from the core ones.

- We can create Paths or Interest Points. Actually, it is the same as enabling the functionality buttons.
- Also it is possible to populate the scene with lots of agents automatically generated. A floating dialog will appear with some parameter to configure the agent properties pseudorandomly. Some boundaries are required to set the possible values for the virtual characters properties.
- *Restart simulation* functionality will put the state of the tool to the initial one. Agent positions, paths, etc.
- The last section is important to facilitate the streaming of Medusa created scenes to other systems. A special section will deal with this functionality, which is under development.

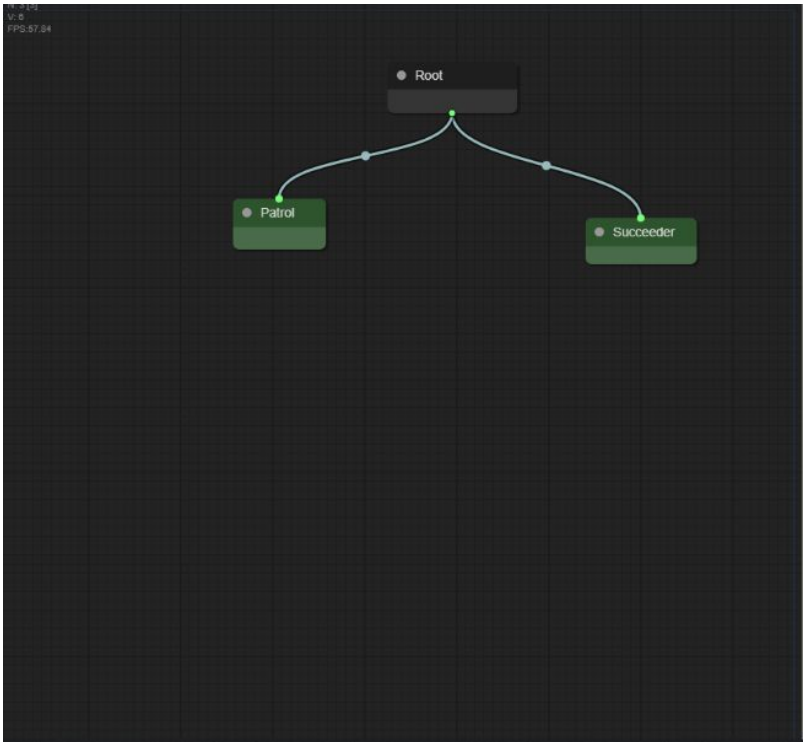
Graph Inspector

This *inspector* is in the upper part of the behaviour creation area, where the graphs are created. Through this inspector we can do several things:

- The first widget starting by the left is the selection of the graph shown in the 2D canvas editor. By default there is only one graph, but in case you are working with more than one graph simultaneously, here is where you set the current one.
- The following button, *New Empty* creates a new empty HBT graph, which means a new behaviour. When you set up the name of the new graph, it is shown directly in the graph editor.

- Next to it, there is the *Load* button. This tool is connected to an open repository of behaviours. Thus, when the load dialog is opened, there will be a list of behaviours available in the repository, where you can find the most suitable for your purpose.
- The *Save* button uploads the current behaviour to the repository. Currently there is no possibility to add more info but the name, but in near future upgrades, it will have the possibility to add tags to the behaviour.
- Finally, the *Download* button. When clicked, it will download a JSON file containing all the current behaviour. To use it, you just have to drag and drop (over the 3D scene and not on the 2D editor due to event capture issues) the file.

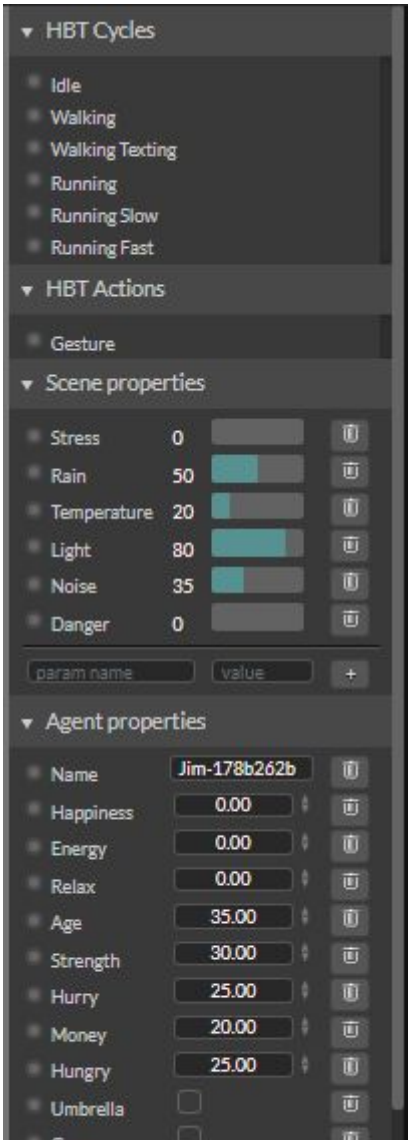
Graph based Editor Area



This area of the interface is where the graphs (behaviours) are created. There are several interactions such as *add*, *connect* and *remove* nodes:

- Add
  - Double click and write the name of the node you are looking for
  - Right click on the canvas, and search for the desired one.
  - Dragging and dropping properties from the scene properties or agent properties. Also dropping the HBT Tasks placed on the top right part of the interface.
- Connect
  - Drag from an output and drop over an input. If an input is highlighted when you are over it, it means that the connection is allowed. Otherwise, you cannot connect the two nodes on that input.
- Remove
  - Right click on the node and Delete
  - On node selected, press the Delete key.

Right Inspector



This part of the interface is where all the properties are placed. The Scene properties, the Agent properties and also the possible Actions related to animations. These are currently only the default ones, but more can be added by dragging and dropping .dae files to the 3D scene.

All these properties are modifiable by changing the values. Moreover, all these properties can be used in the graph editor as HBTProperties acting as input for other nodes. You can use them by dragging the small circle placed in the left of each property and dropping it into the graph.

Hybrid Behaviour Tree Nodes

Common Behavior Tree Nodes

The tick of the tree starts at the root node, which does not have any execution particularity, it only ticks its children. Once the Root is ticked, there are several possibilities:

Conditional Node/BoolConditional Node

When a Conditional Decorator is reached, it evaluates the condition attached to it (editable in the widget embedded), and, if it is true, ticks its children. If it is false, it will return false to its parent, and the parent will determine what to do, depending on its type. It will compare the left input value with the threshold set up.

There are two nodes to distinguish between Conditional and BoolConditional due to its graphical representation of the inner widgets.

Sequencer Node

A Sequencer will tick its children one by one, and if some of them return fail, the return value of this node (the sequencer) will be fail, so its parent will execute the next children in case it has any. In case a task has not finished, but has not failed, the Sequencer saves the child being executed to tick it directly in the next evaluation.

Selector Node

The Selector node works in a similar way as the Sequencer, but it allows children to return false. In case one child returns false, it will tick the next children. It only returns false to its parent if all of its children return false. When a child returns true, this is the branch to follow. Equally, in case a task has not finished, but has not failed, the Selector saves the child being executed to tick it directly in the next evaluation.

LineOfSight Node

Computes the angle between the front direction of the agent and the target where it has to look at, and if the angle is inside a range (set up in the embedded widget) it will succeed. If not it will fail. It has a flow input, where the target evaluated is passed through.

SimpleAnimate

When a SimpleAnimate node is reached, it will return an animation configuration (including the clip to apply, the speed and the motion of the character).

LookAt Node

In this case, this is another leaf node. Thus, it will not compute anything, it will just return the look at position set up by the left input it has. If there is no position (due to any reason, no connection to some flow node, i.e.), it will return false.

SetProperty Node

It changes some parameter to other values of the agent (hurry, mood, if it has something) or even of the scene (if, for example, it turns off the light of all the scene).

Wait Node

When a wait node is executed, it pauses the execution of the branch the time set up. In case a more proprietary branch succeeds, it is reinitialized and this branch gets unblocked.

MoveTo Node:

This node returns a target position for the virtual character evaluated. Also, it will remain on running state until it reaches the target position. It needs to have an input value through the left input link to be executed.

Flow Nodes for the Behavior Tree

These nodes will affect some common BT nodes. For example, they will act as an input for conditionals or for tasks like MoveTo or LookAt. The node will set the property of the agent in a previous execution of this node before the whole Behaviour Tree is evaluated.

EQSNearestInterestPoint

It evaluates which is the nearest Interest Point on the scene for the current virtual character. In the node representations which type of interest point is looking for has to be set. The output is the position of such Interest Point.

EQSDistanceTo

As well as the EQSNearestInterestPoint, it will compute the distance between two given inputs. It can be the distance between the current character evaluated and some interest, the distance between two agents or two interest points, among other possibilities.
It outputs (through the right link) the distance, which could be used for conditionals.

EQSNearestAgent

This node permits to get the position of the nearest virtual character in the scene with a particularity. You have to set a property condition and a value, so this node will find the nearest agent meeting the condition. It will output the position of such a virtual character if it exists.