D5.2 – 3D Simulation environments v1

This project has received funding from the European Union's Horizon Europe research and innovation programme under the grant agreement number 101147722. The European Union is not liable for any use that may be made of the information contained in this document, which is merely representing the authors' view.

Project Acronym:	AccesS
Project Full Title:	Enhancing Accessibility and Sustainability in Smart Cities and Smart Buildings: The Universal Accessibility Suite Initiative
Grant Agreement:	101147722
Project Duration:	36 months (01/06/2024 – 31/05/2027)

DELIVERABLE 5.2
3D Simulation environments v1

Work Package:	WP5 - Virtual User Modeling and Assistive Technologies for Accessibility
Task:	T5.2 - 3D Simulation environnements
Document Status:	Final
File Name:	AccesS_D5.2_3D_Simulation_environments_v1_CERTH
Due Date:	30.11.2025
Submission Date:	28.11.2025
Lead Beneficiary:	CERTH

Dissemination Level	
Public	☒
Sensitive	☐

Authors List

Leading Author (s)				
First Name		Last Name	Beneficiary	Contact e-mail
Vasilis		Sidiropoulos	CERTH	vsidiropoulos@iti.gr
Thomas		Varelas	CERTH	thomvare@iti.gr
Co-Author(s)				
#	First Name	Last Name	Beneficiary	Contact e-mail
1	Thanos	Tsakiris	CERTH	atsakir@iti.gr
2	Georgia	Papadopoulou	CERTH	georgia.papadop@iti.gr
3	Panagiota	Chatzipanagiotidou	CERTH	phatzip@iti.gr
4	Emmanouil	Zidianakis	FORTH	zidian@ics.forth.gr
5	Eirini	Kontaki	FORTH	ekontaki@ics.forth.gr
6	Antonis	Agapakis	FORTH	agantos@ics.forth.gr
7	Michalis	Foukarakis	FORTH	foukas@ics.forth.gr
8	Eva	Fernández Palacios	Voltiva	eva.fernandez@voltiva.energy
9	Raquel	Ortega Martínez	CETEM	r.ortega@cetem.es

Reviewers List

Reviewers			
First Name	Last Name	Beneficiary	Contact e-mail
Laura	Casolo Ginelli	HIVE	laura.casologinelli@hivepower.tech
Harshita	Thakare	LAMA	harshita.thakare@agenzialama.eu
Guido	Van Arkel	BHA	g.vanarkel@bruco.org

Version History

v	Author	Date	Brief Description
0.1	Vasilis Sidiropoulos, CERTH	10.09.2025	Initial draft (ToC)
0.2	All authors, CERTH	17.10.2025	Draft version with initial content for all sections
0.4	All authors	31.10.2025	Updated based on partner's feedback
0.6	Vasilis Sidiropoulos, Thomas Varelas, CERTH	19.11.2025	Final draft for internal review
0.7	Vasilis Sidiropoulos, CERTH	26.11.2025	Updated based on partners feedback
0.8	Panagiota Chatzipanagiotidou, CERTH	27.11.2025	Quality check and minor corrections
1.0	Dimosthenis Ioannidis, CERTH	28.11.2025	Available for submission to the EC

Legal Disclaimer

The AccesS project has received funding from the European Union's Horizon Innovation Action under grant agreement No 101147722. The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Climate, Infrastructure and Environment Executive Agency (CINEA) or the European Commission (EC). CINEA or the EC are not responsible for any use that may be made of the information contained therein.

Copyright

Executive Summary

This deliverable presents the first version of the 3D Simulation Environments developed within the AccesS project. These environments form a central component of the project's accessibility simulation toolkit, enabling realistic, data-driven evaluation of how diverse Virtual User Models (VUMs) interact with buildings and urban spaces. The work documented here establishes the technological foundation required for producing consistent, semantically rich, and simulation-ready representations of built environments, and for combining them with VUMs to execute detailed accessibility assessments.

The report introduces the complete workflow implemented in this first version: the preparation of 3D environments, the annotation of interactable elements, the export of geometry and semantic metadata, the construction of simulation scenarios, and the execution of physically and behaviourally grounded simulations. It outlines the design and implementation of the SANTO annotation tool, which translates Industry Foundation Classes (IFC) derived building models into interactive virtual environments through a structured process of semantic enrichment, parametrisation of movement affordances, and validation. The output of this workflow consists of paired geometric (.glb) and semantic (.json) files, ensuring interoperability and reproducibility across simulation modules.

Building on these environments, the deliverable details the integration of the Scenario Generator, which allows users to define step-wise behaviours, select VUMs, and run simulations that assess accessibility challenges in practical contexts. The report documents the underlying algorithms for kinematic planning, motion generation, inverse dynamics analysis, and the simulation of sensory impairments such as visual and auditory challenges. These components together enable the generation of physically coherent avatar movements and provide quantitative indicators of accessibility barriers, including joint loading, required interaction forces, or visibility constraints.

Initial testing and validation activities are also presented. These include unit and integration tests, communication with the external VUMS platform, and preliminary verification of simulation accuracy through comparisons with published biomechanical data. Early findings demonstrate strong alignment with known gait patterns in simple scenarios and highlight specific challenges to be addressed in stair-related movements, informing the roadmap for the next release.

Overall, this first version offers a robust and modular framework for creating and simulating interactive environments that reflect the needs and limitations of diverse user profiles. It sets the stage for the upcoming iterations, where refinements of annotation workflows, motion algorithms, and validation procedures will support the delivery of a fully mature accessibility simulation platform.

Table of Contents

List of Figures

List of Acronyms and Abbreviations

Term	Description
API	Application Programming Interface
BIM	Building Information Modeling
CAD	Computer-Aided Design
D	Deliverable
GLB	Binary version of the GL Transmission Format (3D model format)
glTF	GL Transmission Format (3D model format)
HCI	Human-Computer Interaction
IFC	Industry Foundation Classes (open BIM standard)
IK	Inverse Kinematics
JSON	JavaScript Object Notation
UI	User Interface
VUM	Virtual User Model
VH	Virtual Human
WP	Work Package

1 Introduction

1.1 Scope and objectives of the deliverable

This deliverable describes the accessibility simulation platform developed within T5.2. The core components and the core architecture are described in detail alongside the first implementations that developed for the initial testing and validation of the platform. There are two main subsystems, the simulation environments and the simulation creation. The simulation environment subsystem enables users to create the simulation environments, using 3D models of buildings they want to execute accessibility simulations for. The simulation creation subsystem is the core simulation application where all other components namely the Virtual User Models, the simulation environments and the simulation scenarios are processed in order to execute an accessibility simulation and output the assessment. In each section, the core components and functionalities of each subsystem is discussed.

1.2 Structure of the deliverable

This deliverable is organised to follow the logical flow of creating, annotating, and using 3D environments for accessibility simulations, starting from the conceptual overview and progressing toward implementation, validation, and outcomes. The document is divided into five main sections.

- **Chapter 1** introduces the purpose and objectives of the deliverable and situates it within the broader context of the AccesS project;
- **Chapter 2** presents an overview of the 3D simulation environment architecture;
- **Chapter 3** presents the 3D Simulation Environment subsystem. This part describes how building models are imported, inspected, and semantically annotated, and how interactive elements are defined using the 3D Environment Generator tool. It also provides the foundational information required for understanding how simulation environments become “simulation-ready” assets used in later stages;
- **Chapter 4** focuses on the Simulation Scenario Generator, which consumes the environments produced in Section 2. It explains how users configure simulation steps, select Virtual User Models, and execute accessibility simulations;
- **Chapter 5** addresses the testing and validation of the system. It outlines the unit and integration testing performed during development;
- **Chapter 6** concludes the document by summarising the current status of the 3D simulation platform and outlining the planned improvements for the next version.

1.3 Relation to Other Tasks and Deliverables

Deliverable D5.2 is directly rooted in the work of Task T5.2, where the 3D simulation environments are designed and implemented for assessing VUM-based Virtual Humans. Its development depends closely on the outcomes of Task T5.1, which generates the Virtual User Models and the corresponding Virtual Humans, as these models constitute the primary input for testing within the simulation environments. The VUM schema editor and VH toolchain delivered in T5.1 provide the modelling and authoring capabilities required before the simulations can be executed, establishing a direct workflow link between the two tasks and corresponding deliverables.

The synthetic and reconstructed 3D environments produced in T5.2 also contribute to later deliverables in WP5, specifically D5.4, which expands these environments with additional content and sensitive information for internal use. Beyond WP5, the outputs of D5.2 are essential enablers for several technical WPs. Most notably, they may support WP7, where simulation-based testing is

required for the Large-Scale Accessibility Assessment Tool to be delivered in T7.1 while it also linked with the integrated Suite and testing in T7.2 and T7.3.

The environments created are also relevant for downstream demonstration activities in WP9–WP11, as they help validate and refine designs before physical pilot deployments. Furthermore, the completion of D5.2 directly contributes to Milestone 4 – Simulation with VUM-based Virtual Humans enabled. In summary, D5.2 acts as a bridge between user modelling (T5.1), assessment and integration tooling (WP7), and pilot-oriented validation (WPs 9–11). Its role is foundational in enabling simulation-driven evaluation of accessibility across the AccesS digital ecosystem, ensuring coherence between model creation, simulation, and subsequent large-scale assessment.

2 Overview

The developed 3D scenario simulator (Figure 1, green color) comprises two main components the 3D environment generator and the Scenario Generator while there are also two integration managers for the environments and the VUM integrations respectively, as described within D1.2 AccesS Framework architecture, use cases and master planning for demonstrators.

The 3D environment generator enables users to construct virtual environments that serve as the basis for subsequent simulations. Within this application, users can import three-dimensional (3D) building models intended for analysis and configure their interactive components. Upon completion, the created environment can be exported and be used later in the Scenario Generator.

The Scenario Generator integrates input from both the environment preparation application and the VUM editor (D5.1). Users can define and execute simulation scenarios within the application. During execution, the accessibility simulation runs a multilevel simulation and generates a detailed output assessment report summarizing the results.

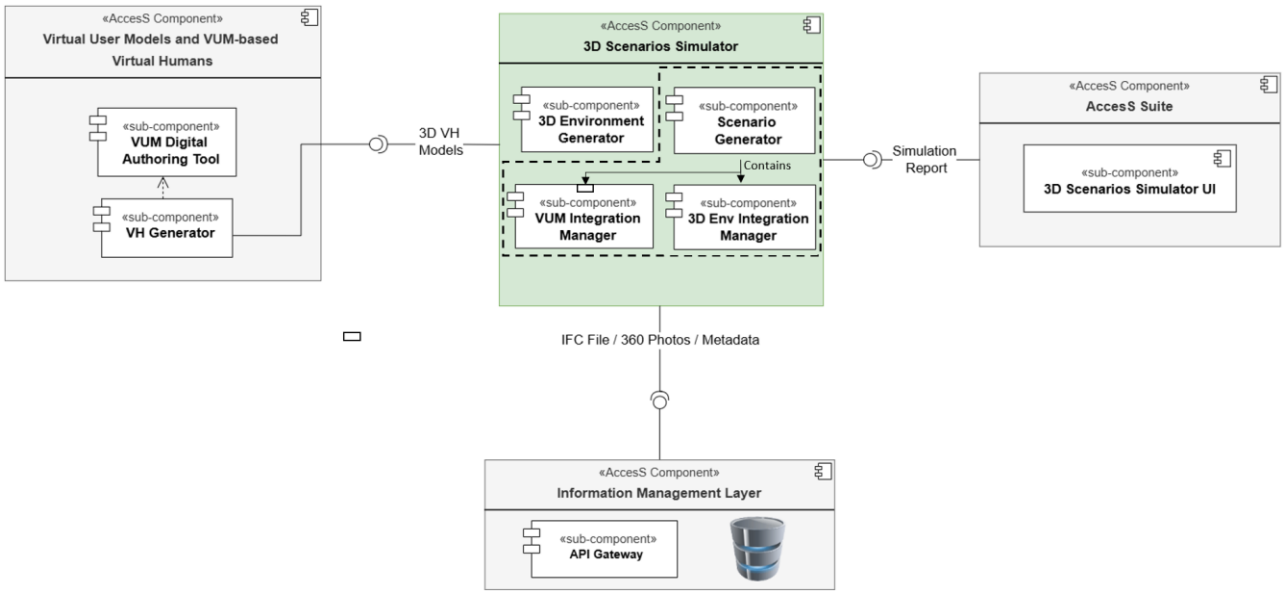

Figure 1. Component Diagram of 3D Simulation Engine

3 3D Environment Generator

3.1 Purpose and role in the workflow

3D Simulation Environments serve as a critical translation layer for the assessment of the generated VUM-based Virtual Humans (VHs). These environments can be either synthetic implemented using 3D modeling and authoring tools, or digital replicas of actual environments (through a combination of aerial and terrestrial 3D reconstruction technologies) in alignment with the pilot cases of the project. To this end, a tool developed to semantically annotate 3D environment objects (SANTO) that VHs may interact with, such as doors, switches, or drawers, enabling more realistic perception and interaction modeling in virtual simulations. Simulation results derived from these annotated environments support the identification of design considerations that enhance the development of inclusive and adaptable solutions designed for all users.

3.2 User interface overview

The iterative design methodology structured the UI of the SANTO around a continuous cycle of development, evaluation, and refinement. The UI design of the SANTO reflects well-known established principles of Human-Computer Interaction (HCI) suitable for complex parametric systems. It allows users to efficiently identify, isolate, and manipulate specific objects in the 3D viewport from a potentially dense architectural model (see Figure 2.) which embodies the crucial HCI principle of Direct Manipulation [46]. This approach is essential for providing users with immediate visual confirmation and focusing their attention on the precise element being modified, thereby reducing cognitive load and errors in property management [17].

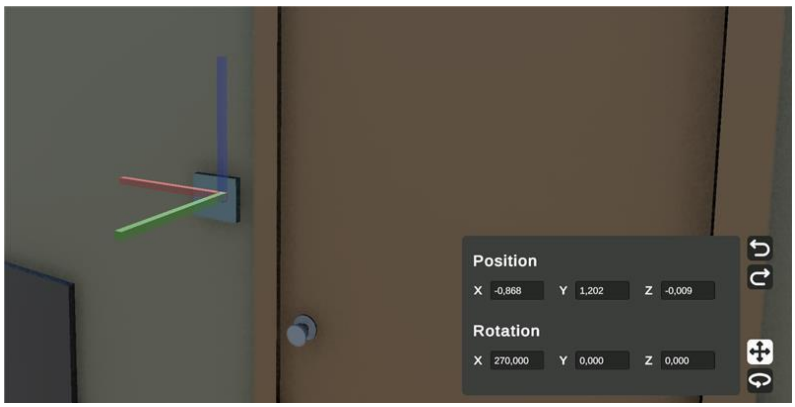

Figure 2. SANTO's Gizmo interface for manipulating physical objects

3.3 Object Hierarchy User Interface

As depicted in Figure 3., the Hierarchy User Interface is displayed on the left side of the screen. This interface presents a structured list of all physical objects within the scene, preserving the inherent parent-child relationships defined in the Unity engine. Each visible object is represented by a unique entry containing its identifier (ID). Objects containing subordinate elements include an expandable dropdown control, allowing users to display or conceal the corresponding child objects.

In addition to the object identifiers, specific icons are displayed alongside each object name, representing the corresponding IFC category. These icons provide both class-type information and annotation status. The shape of the icon reflects the object's IFC classification, while the color indicates its annotation state as follows:

- Green icons denote objects that have already been annotated;
- Yellow icons indicate objects that have not yet been annotated.

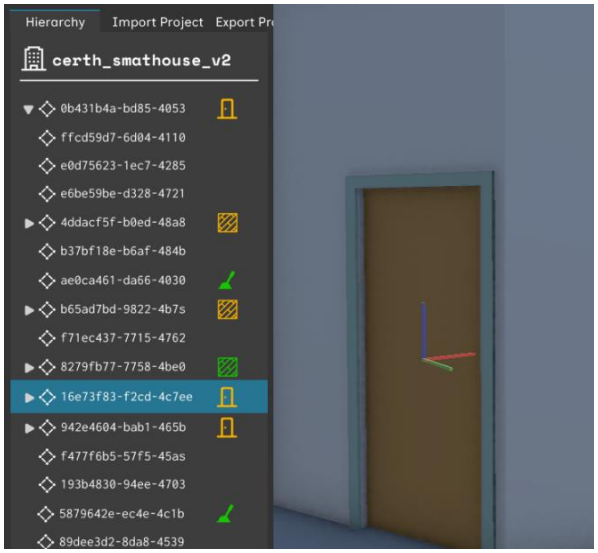

Figure 3. The Hierarchy UI.

This visual enhancement significantly improves the readability and usability of the interface, enabling users to efficiently assess the annotation progress across the environment and to identify objects that require further processing.

3.4 Annotation UI

When a non-annotated object is selected from the hierarchy, the *Inspection Panel* is displayed on the right side of the screen. This panel presents key information about the selected object, including its name, and provides a dedicated button to initiate the annotation process (see Figure 4.).

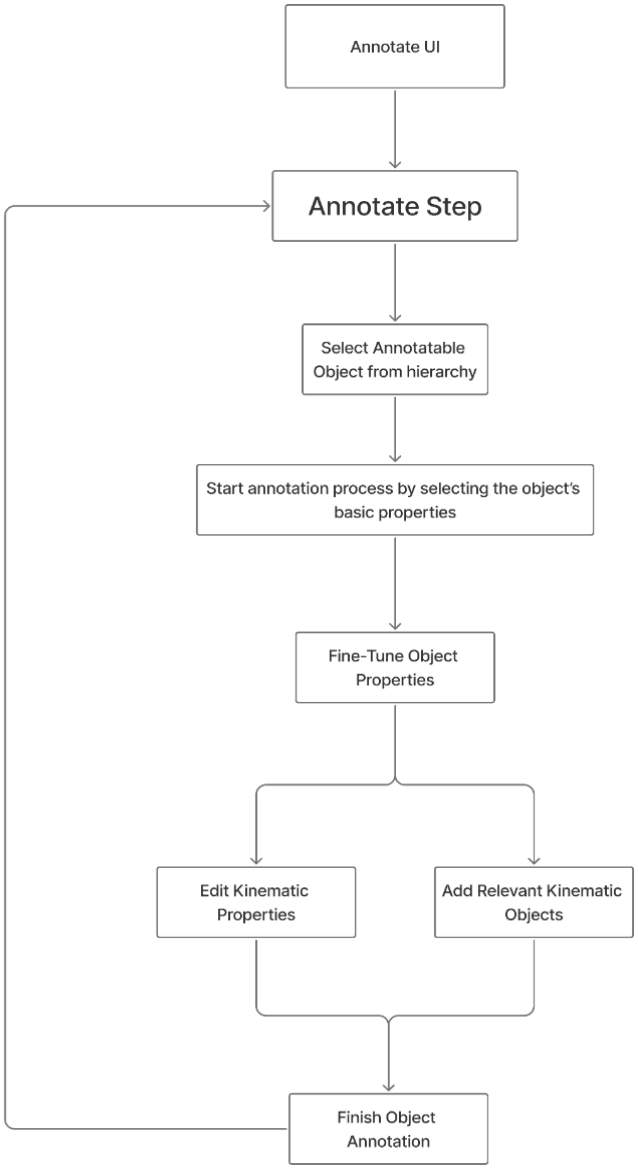

Figure 4. Overview of the Annotate Step workflow.

Upon activation, the annotation workflow begins. The annotation procedure is adaptive and varies according to the object type, as defined by its associated IFC metadata. For instance:

- In the case of floors, a simple annotation is sufficient, typically involving the assignment of a box collider;
- For more complex elements, such as doors, the annotation requires a multi-step process, which will be described in detail in section 3.6.

This modular approach ensures that the annotation mechanism remains flexible and context-aware, supporting efficient handling of diverse object types within the digital environment.

Upon selection of an annotated object (either from the hierarchy or by clicking its object in the 3D scene), an editing panel (see Figure 5.) appears on the right, displaying all relevant information. Contextual menus are vital for enhancing usability by offering options that are highly relevant to the user's current task and the selected object type (e.g., exposing Kinematics for a door but hiding it for a wall) [18]. Housing less frequently used or complex configuration options, such as detailed kinematic

settings, within this context-sensitive panel effectively reduces overall UI clutter, ensuring the primary visualization and manipulation features remain accessible [18].

Objects can be selected either directly from the hierarchy panel or by interacting with them within the 3D scene. Navigation within the 3D environment is achieved using standard, for 3D editing software, mouse controls: holding the right mouse button enables rotation, holding the middle mouse button allows panning, and zooming is performed via the mouse scroll wheel.

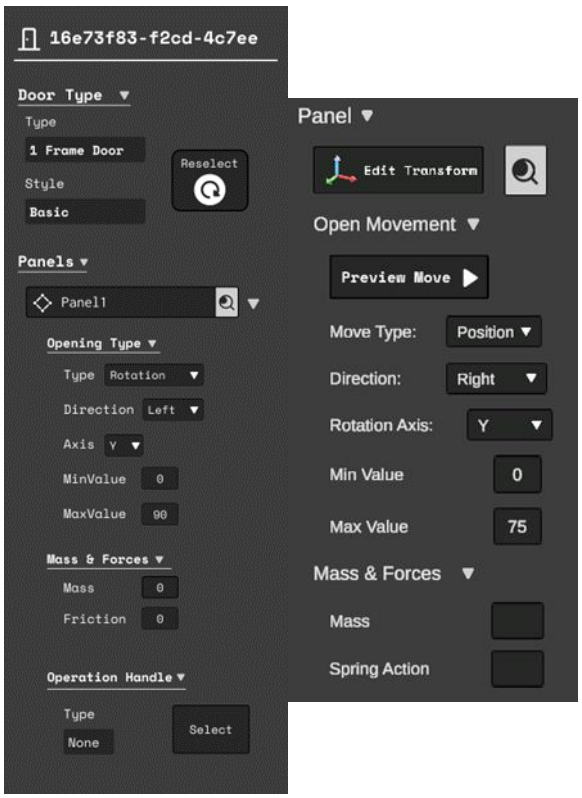

Figure 5. Object Properties Editing Panels

The methods for viewing annotated objects are interactive and translate the often-complex querying methods found in BIM systems, such as filtering elements using logical expressions like *"$type = wall"* and *"Height > 5000"* in tree-structure panels, into an intuitive point-and-click interaction, backed by the model's semantic metadata structure [19].

3.5 Defining interactable elements

To realize a true simulation environment, the *SANTO* system must translate the passive, structural semantics of the IFC object into an active, executable schema that dictates physical interaction. This translation is formalized within a custom JSON structure.

3.5.1 Extraction and Mapping of IFC Object Semantics

The preparatory phase involves extracting critical data from the imported IFC file, focusing specifically on elements identified as motion-bearing components [27]. For example, for each object with the class *IfcDoor*, the system initially retrieves static properties such as *Width*, *Height*, *Thickness*, *Door Material*, and crucial semantic tags like *Function* (Interior/Exterior) [6].

The IFC schema provides semantic clues regarding movement potential. For example, the *IfcDoorStyle* contains *IfcDoorPanelProperties*, which define parameters related to the door's operational mechanism, such as PanelOperation (Swing, Sliding) [28]. These semantics serve as the initial input to determine the appropriate Unity physics constraint (e.g., a Hinge Joint for a swinging door) and applicable motions which the user may parametrize. Necessary pre-processing includes automated classification to map the extracted BIM elements to the simulation schema [29].

3.5.2 Standardization of Kinematic Affordances for Avatar Interaction

For creating 3D environments that are suitable for realistic and accurate simulations, a requirement is to establish a standardized representation of kinematic affordances for interactable objects within the virtual 3D environment. Kinematic affordances define the possible actions a human avatar or simulated agent can perform with an object (e.g., rotation, translation, or resistance-based interactions).

Standardization of this data ensures that the exported JSON structures are universally interpretable and reusable by downstream simulation platforms, thus enabling interoperability between modules developed within and beyond the project.

To collect, control, and parameterize the kinematic affordances of interactable objects, a set of internal data structures and classes has been implemented. These structures enable the consistent definition of object behaviors related to physical interaction and motion.

The AccessMovementType enumeration specifies the type of movement an object can perform and includes three possible values:

- None: the object does not support any movement;
- Position: the object can be translated along an axis;
- Rotation: the object can rotate around a defined axis.

The AccessDirection enumeration defines the direction of movement, taking the values Right, Left, Up, Down, Forward, Backward, and None, covering the full range of motion within a three-dimensional coordinate space.

The AccessAxis enumeration identifies the axis along which the movement occurs (X, Y, Z, or None).

All of these enumerations are encapsulated within the AccessMovement class, which acts as a comprehensive data container for an object's kinematic affordance. In addition to the enumerations, this class defines several quantitative parameters:

- fresist: a floating-point value representing the resistance force that must be overcome for the movement to occur;
- springAction: a value expressing the restoring force that returns the object to its initial state following interaction;
- minValue and maxValue: numerical limits specifying the range of allowed movement.

The AccessMovement class therefore encapsulates both IFC-derived and user-editable parameters. While some fields remain immutable to safeguard data integrity and standardization, others are configurable through the user interface, allowing controlled adjustments to the behavior of specific objects.

The aforementioned enums and structural base classes are depicted in Figure 6. while a comprehensive implementation example of the AccessMovement class and its usage is detailed in section 3.6.

```
public enum AccessMovementType
{
    None,
    Position,
    Rotation
}

public enum AccessAxis
{
    X,
    Y,
    Z,
    None
}
public enum AccessDirection
{
    Left,
    Right,
    Up,
    Down,
    Forward,
    Backward,
    None
}

public class AccessMovement
{
    public AccessMovementType type;
    public AccessDirection direction;
    public AccessAxis axis;
    public float fresist;
    public float springAction;
    public float minValue;
    public float maxValue;
}
```

Figure 6. SANTO enums and structural base classes

3.5.3 Semantic Representation Schema

A standardized semantic representation schema in JSON format formalizes the interaction metadata exchanged between the simulation environment and the simulation engine. This schema serves as the interoperability contract ensuring that all interaction logic, object hierarchies, kinematic parameters, and state definitions are machine-readable, consistent, and reusable across different software modules and simulation frameworks.

The schema follows established standards such as JSON Schema Core, while maintaining a modular structure that aligns semantic, geometric, and behavioral data. Each JSON instance links directly to the corresponding 3D geometry (GLB) and encodes object-specific interaction parameters in a form suitable for downstream physics or AI-based simulation.

3.5.3.1 Interaction Metadata

The interaction metadata JSON describes how physical objects in the 3D environment, such as doors, handles, and panels, behave when interacted with by an avatar or agent. It captures both structural relationships (through object hierarchies) and functional properties (through movement definitions and state logic).

The purpose of this structured JSON is to ensure a unified and interpretable data model where interactive objects can be reconstructed, simulated, or visualized consistently across environments. The key components are described below, following the structure illustrated in the example JSON for the *Single Panel Door* as depicted in Figure 7..

```
{
   "object_id": "door_001",
   "type": "door",
   "tag": "Single Panel Door",
   "children": [
      {
         "object_id": "panel_main",
         "type": "panel",
         "tag": "Main Door Panel",
         "movements": [
            {
               "movement_origin": "-0.40,0,0",
               "movement_axis": "0,1,0",
               "movement_type": "rotation",
               "min_value": 0,
               "max_value": -95,
               "fresist": 45,
               "spring_action": 0.0
            }
         ],
         "children": [
            {
               "object_id": "handle_main",
               "type": "handle",
               "tag": "Door Handle",
               "movements": [
                  {
                     "movement_origin": "0,0,0",
                     "movement_axis": "0,0,1",
                     "movement_type": "rotation",
                     "min_value": 0,
                     "max_value": 95,
                     "fresist": 45,
                     "spring_action": 0.5
                  }
               ],
               "states": []
            }
         ],
         "states": []
      }
   ],
   "states": [
      {
```

```
      "name": "panel_open",
      "conditions": [
         {
            "object_id": "panel_main",
            "value_type": "rotation",
            "axis": "0,1,0",
            "valueTests": [
               {
                  "comparison": "greater_than",
                  "value": "90"
               }
            ]
         }
      ]
   },
   {
      "name": "panel_closed",
      "conditions": [
         {
            "object_id": "panel_main",
            "value_type": "rotation",
            "axis": "0,1,0",
            "valueTests": [
               {
                  "comparison": "less_than",
                  "value": "10"
               }
            ]
         }
      ]
   },
   {
      "name": "handle_main_open",
      "conditions": [
         {
            "object_id": "handle_main",
            "value_type": "rotation",
            "axis": "0,1,0",
            "valueTests": [
               {
                  "comparison": "greater_than",
                  "value": "90"
               }
            ]
```

```
            }
         ]
      },
      {
         "name": "handle_main_closed",
         "conditions": [
            {
               "object_id": "handle_main",
               "value_type": "rotation",
               "axis": "0,1,0",
               "valueTests": [
                  {
                     "comparison": "less_than",
                     "value": "10"
                  }
               ]
            }
         ]
      },
      {
         "name": "panel_half-open",
         "conditions": [
            {
               "object_id": "panel_main",
               "value_type": "rotation",
               "axis": "0,1,0",
               "valueTests": [
                  {
                     "comparison": "greater_than",
                     "value": "40"
                  },
                  {
                     "comparison": "less_than",
                     "value": "50"
                  }
               ]
            }
         ]
      }
   ]
}
```


Figure 7. Door Semantic Annotation Example

3.5.3.2 Identification Fields

Identification fields establish the link between the interaction metadata and its corresponding geometric representation in the 3D model. The primary field, object_id, directly references the GameObject's unique identifier as exported from the Unity engine in .glb format. This allows the simulation engine to bind each JSON entry to its exact mesh node within the 3D scene hierarchy. In addition to the object_id, the fields type and tag provide semantic and organizational metadata:

- type defines the object's classification (e.g., "door", "panel", "handle"), facilitating filtering, automated behavior assignment, and logical grouping;
- tag provides a human-readable descriptor, supporting interpretability and debugging during development and testing.

These fields collectively support data organization, hierarchical sorting, and consistent classification, ensuring that multiple simulation components can interpret the same object structures in a unified way.

3.5.3.3 Kinematic Definitions

The kinematic definitions describe the object's potential movements and mechanical affordances. Each object can include one or more entries within the movements array, defining the nature and constraints of its possible motions.

A movement definition includes:

- movement_origin: the pivot or effector point (in 3D coordinates) from which the motion originates;
- movement_axis: the axis vector describing the direction or rotational axis of movement;
- movement_type: the motion category, which can be either *rotation* or *translation*;
- min_value and max_value: the range limits constraining the amount of motion (e.g., rotational degrees or translation distance);
- fresist: the resistance force representing the mechanical effort required to move the element;
- spring_action: the restoring force coefficient simulating the natural tendency of an object to return to its original state (for example, a self-closing door).

These kinematic parameters correspond directly to the AccessMovement class described in section 3.5.2, ensuring a consistent data structure between the UI annotation layer and the serialized metadata used in simulation.

In the provided example, the panel_main entry defines a rotation of - 95 degrees around the Y - axis, representing the door's opening movement. Similarly, the handle_main entry defines a separate rotation around the Z-axis, describing the turning motion of the handle.

3.5.3.4 State Logic

The state logic defines the discrete, named conditions of an object based on measurable kinematic parameters, such as rotation angles or translation distances. States allow the system, or the annotating expert, to express functional interpretations of numerical motion data (e.g., *open*, *closed*, *half-open*).

Each entry in the states array contains:

- a name field identifying the logical state (e.g., "panel_open");
- a list of conditions, each specifying an object_id, value_type (e.g., rotation), and one or more valueTests.

Each valueTest applies a comparison rule (e.g., "greater_than" or "less_than") to evaluate whether the object's current kinematic parameter falls within a defined range. When the conditions of a state are met, the simulation engine can trigger corresponding visual or behavioral responses, such as animations, sound cues, or physical constraints.

In the example schema, the state "panel_open" is semantically enabled when the door's panel rotation exceeds 90 degrees, while "panel_closed" is enabled when the rotation remains below 10 degrees. Similarly, the handle defines its own "open" and "closed" states independently.

This mechanism enables semantic annotation of motion and enhances the contextual understanding of interactions within the simulation environment.

3.5.3.5 Hierarchical Nesting

The schema preserves the parent–child relationships of physical objects using the children array, reflecting their structural dependencies within the 3D scene.
For instance, in the provided example, the door (door_001) contains the main panel (panel_main), which in turn contains the handle (handle_main).

This hierarchical organization offers multiple advantages:

- It mirrors the real-world composition of objects, ensuring natural and intuitive behavior propagation during simulation (e.g., the handle follows the movement of the door panel);
- It enables efficient scene management, as child objects inherit transformations and visibility from their parent;
- It provides a clear data parsing structure, allowing simulation engines or external systems to traverse, index, and manipulate the interaction metadata consistently.

This approach ensures that the simulated environment maintains both structural fidelity and functional clarity, supporting scalable and interoperable modeling of complex interactive assemblies.

3.6 Environment validation and export process

The Environment Validation and Export Process constitutes the final stage in the pipeline, where annotated and parametrized BIM elements are transformed into standardized, simulation-ready assets (see Figure 8.). This step ensures that both semantic accuracy and geometric integrity are preserved, while producing outputs that are efficiently consumable by downstream real-time simulation platforms.

The export process results in two complementary artifacts:

1. A *.glb* file, representing the geometric, visual, and material structure of the environment;
2. A *.json* file, containing all associated interaction metadata, kinematic definitions, and state logic described in section 3.5.3.

Together, these two files form a complete and interoperable representation of the annotated simulation environment.

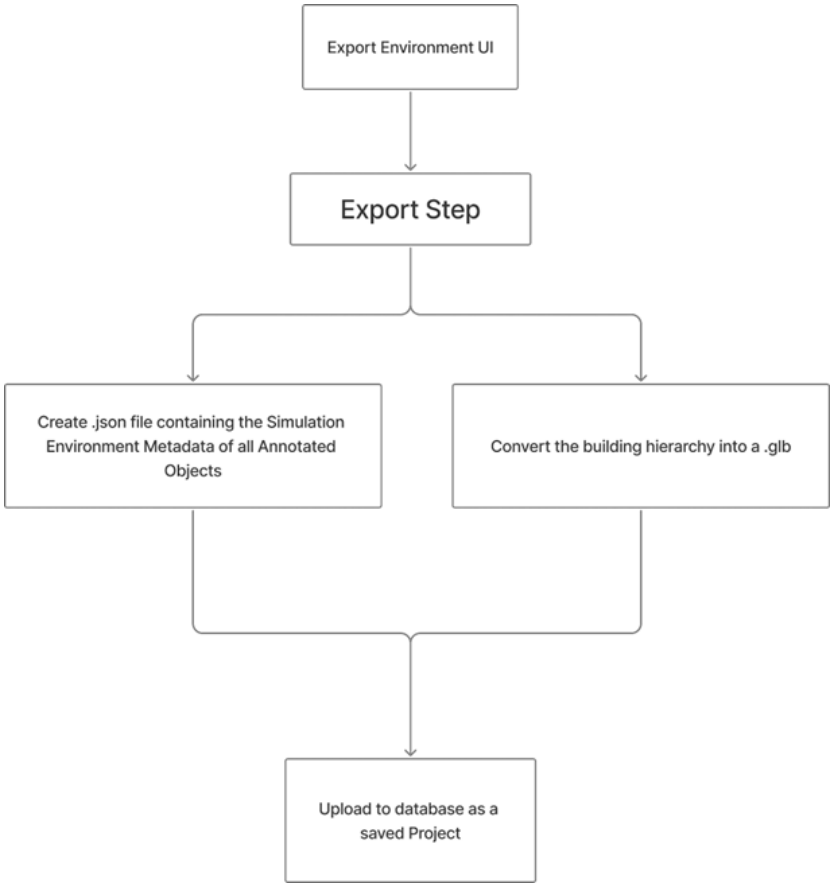

Figure 8. Overview of the Export Step workflow

3.6.1 Pre-Export Validation

Before initiating the export, the system performs a validation procedure to verify annotation completeness and data consistency.

The Hierarchy User Interface provides visual feedback on annotation status through icon coloration: green icons represent annotated objects, while yellow icons indicate objects still pending annotation. If unannotated objects are detected, the system issues a warning prompt, ensuring that the user can address any incomplete definitions before export.

3.6.2 Geometry Export (.glb)

The *geometry export* stage produces a binary *.glb* file from the Unity GameObject hierarchy corresponding to the annotated scene. The export process employs the Unity glTFast library, selected for its adherence to the glTF 2.0 specification and its ability to efficiently serialize complex scenes into compact binary assets suitable for real-time rendering.

The GLB format is preferred over the text-based .gltf equivalent due to its single-file structure, which simplifies distribution, integration, and loading in downstream systems [12]. This binary format encapsulates the entire 3D model, including mesh geometry, textures, materials, and Physically Based Rendering (PBR) definitions, within a single, portable artifact.

A key requirement of this stage is to preserve the hierarchical scene graph of the Unity environment [11]. Each interactable object is exported as a distinct node with a unique identifier corresponding to its original Unity GameObject. This hierarchical structure is essential for maintaining parent–child

relationships and enables precise linkage between the geometric elements in the GLB file and their corresponding interaction definitions in the JSON metadata.

3.6.3 Semantic Metadata Export (.json)

Following the *geometry export*, the environment generates the interaction metadata file (.json), which formalizes the behavioral, kinematic, and semantic information of all annotated objects.

During this stage, data from the internal annotation classes (e.g., AccessMovement, AccessDirection) are serialized into structured JSON entries. Each object's unique identifier (object_id) is used to bind the metadata entry to its geometric counterpart in the exported GLB file. This ensures one-to-one correspondence between visual geometry and interactive behavior.

The JSON file contains a top-level reference to the exported GLB file, enabling simulation engines or consuming applications to automatically load the correct geometry alongside its metadata [34]. Each subsequent JSON entry defines an object's properties (including movement axes, physical constraints, affordance parameters, and state logic) allowing the simulation system to reconstruct the annotated environment dynamically and accurately.

The JSON export process enforces internal validation rules to ensure that:

- All required fields (e.g., *object_id*, *movement_type*, *axis*, *state conditions*) are populated;
- No orphaned references exist (i.e., every JSON object corresponds to a valid node within the GLB hierarchy);
- Editable fields are properly constrained according to IFC-derived semantics and validated user inputs.

This approach ensures that the interaction metadata remains both semantically accurate and structurally consistent, maintaining traceability between the original BIM source and the final simulation environment.

3.6.4 Output and Database Integration

Upon successful validation and export, the resulting GLB and JSON files are combined into a unified simulation environment package. This package is then uploaded to the project's central database of simulation environments.

This repository enables:

- Centralized management and retrieval of simulation-ready assets;
- Version control and traceability of annotated models;
- Reusability of environments by modifying only the JSON logic without re-exporting geometry.

The decoupled architecture, separating geometry (.glb) from interaction logic (.json), provides flexibility for iterative development. For example, updates to motion parameters or affordances can be applied by editing the JSON file alone, without requiring a full re-export of large and complex building geometries.

3.7 Example workflow

Figure 9. illustrates the complete workflow for annotating a door object within the simulation environment, demonstrating how static IFC data is semantically enriched and parametrically configured to achieve a *SimReady* state. The process exemplifies the integration of hierarchical object management, parametric editing, and kinematic affordance definition within the annotation interface.

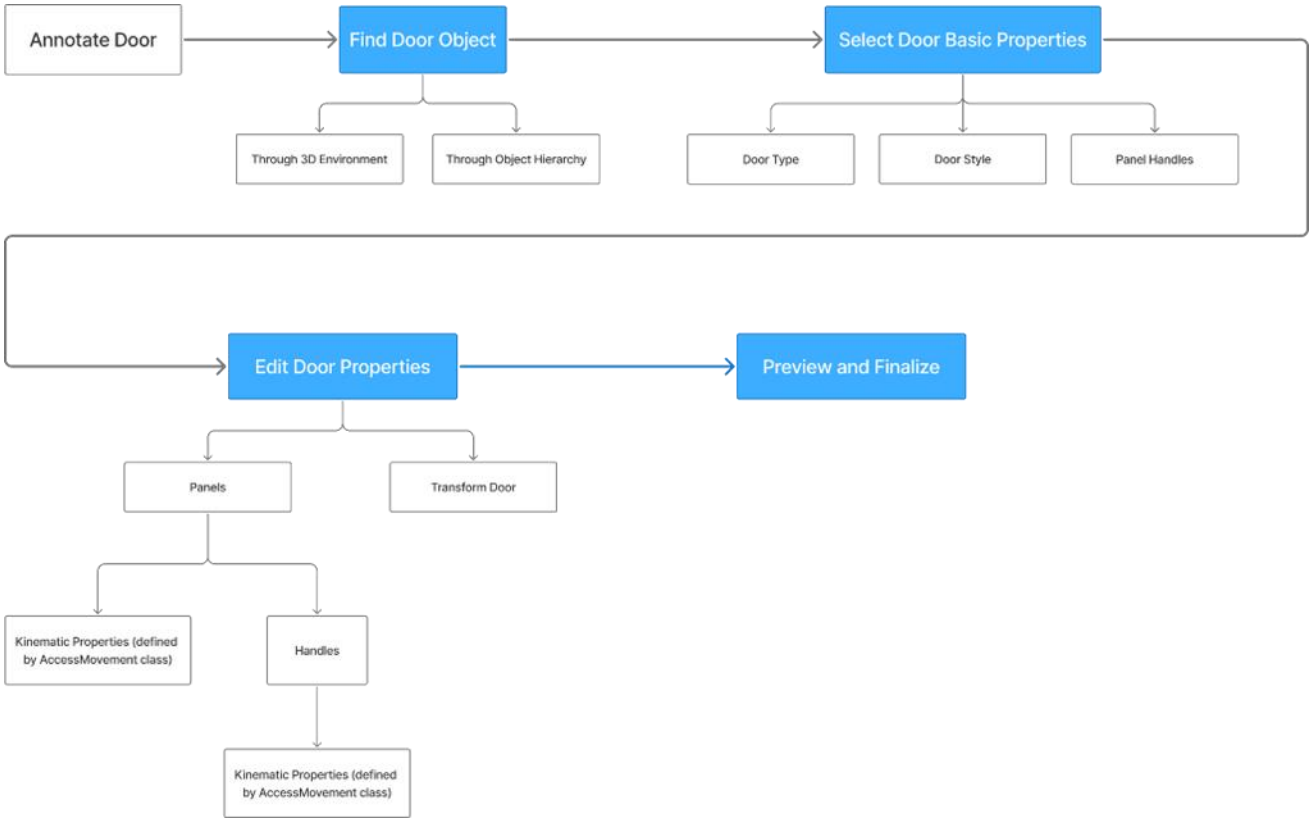

Figure 9. Door Annotation Example Workflow

3.7.1 Step 1: Locating the Door

The annotator begins by identifying the target object within the environment. This can be achieved either by selecting the door directly within the 3D scene or by locating it through the Hierarchy panel.

The hierarchy interface provides visual cues for rapid identification: each object is represented by an icon corresponding to its IFC classification, while its color indicates annotation status (green for annotated objects and yellow for unannotated ones). This visual feedback enables annotators to quickly assess the completion state of all interactable objects within the model.

3.7.2 Step 2: Selecting the Object and Initiating Annotation

Once the user selects the unannotated door, an Annotation Panel appears on the right side of the interface. This panel displays the object's basic information (such as name and identifier) and includes an "Annotate" button to begin the process. Upon activation, the right-side panel transitions into a generation interface, and an auxiliary panel appears in the center of the screen. This central interface guides the annotator through a structured, step-by-step process to define the door's physical and semantic properties.

3.7.3 Step 3: Defining Door Properties

The annotation process begins with the definition of the door's key characteristics (see Figure 10.10). These parameters ensure that the generated simulation model reflects both the intended physical behavior and the contextual accuracy of the environment.

a. Door Type Selection: The first step involves choosing the door's panel configuration - either *single-panel*, *double-panel*, or *triple-panel*. This step accommodates potential discrepancies between the IFC

model and the real-world environment, such as outdated or inconsistent schema definitions. By allowing the annotator to explicitly redefine such parameters, the system ensures that the resulting model maintains physical and logical coherence (see Figure 10.10 left).

b. Door Style Selection: The next step focuses on the visual style of the door. Four representative door styles are available, enabling the annotator to select the one that best matches the architectural and contextual characteristics of the environment. This choice affects only the visual representation and not the functional behavior (see Figure 10.10 center).

c. Handle Type Selection: Finally, the annotator specifies the handle type for each door panel. Available options include (see Figure 10.10 right):

- Lever Handle
- Knob Handle
- Push/Pull Handle
- Button Handle (for electronic or elevator doors)
- None (for doors without handles)

Figure 10. The panels for the selection of Door Properties: DoorType (left), DoorStyle (center), Panel Handle (right)

After confirming these selections, the system replaces the original static IFC-derived model with a newly instantiated simulation ready door object. The new model preserves the original dimensions and transform while incorporating the defined visual and interactive characteristics.

3.7.4 Step 4: Editing Door Parameters

Following annotation, the right-side Edit Panel is populated with the door's detailed parameters, allowing the annotator to review or refine its configuration.

3.7.4.1 Door UI Content

The main door properties interface includes:

- The object identifier (ID) displayed as the panel's title.
- A display of the selected style and type from the previous annotation step.
- A "Restart Annotation" button allowing the user to reinitiate the selection process (type, style, handle).
- A "Focus on Object" button to center the 3D viewport on the door.
- An "Edit Transform" function enabling precise translation, rotation, and scaling of the object via a 3D gizmo or direct numeric input.

Although the generated simulation ready door automatically inherits the IFC object's position, rotation, and scale, the availability of performing manual adjustments ensures that it aligns perfectly within the simulated spatial context.

Below these general controls, the *Panel UI* is displayed for each door panel contained within the parent door entity.

3.7.4.2 Panel UI Content

Each door panel, following the hierarchical IFC paradigm, is treated as a distinct subcomponent with its own editable properties.

For each panel, the interface provides:

- A "Focus" button to isolate and view the panel in the 3D environment.
- An "Edit Transform" option, similar to that of the parent door, for fine-tuning its spatial alignment.
- An expandable "Open Movement" UI sub-component containing the kinematic parameters defined by the corresponding AccessMovement class (see section 3.5.3). This UI sub-component includes editable fields for:
    - Movement Type (Position or Rotation)
    - Direction (Left, Right, Up, Down, Forward, Backward)
    - Axis (X, Y, Z)
    - Minimum and Maximum Values (defining the movement range)
    - fresist (resistance force)
    - springAction (restoring force coefficient)

These parameters collectively define the physical constraints and behaviors associated with the door's motion.

A "Preview Move" button is provided, allowing the user to visualize the movement defined by the current parameters. This action performs a non-simulated motion within the environment—serving purely as a visual verification tool to confirm that the rotation axis, direction, and range of motion align with the intended behavior (e.g., confirming that a door swings open correctly).

Each panel section also includes a Handles UI sub-component, which lists all handle components associated with that panel. Each handle includes its own AccessMovement definition and user interface, allowing for fine-tuning of its interaction properties (for instance, the rotation of a lever or the press depth of a button).

3.7.5 Step 5: Final Verification and Completion

After reviewing or adjusting the kinematic and positional parameters, the annotator can preview the complete interaction to verify correctness. Once satisfied, they can proceed to annotate additional objects or finalize the current environment for export.

This workflow ensures that each door, initially a static IFC entity, is systematically transformed into a fully defined, semantically consistent, and simulation-ready interactive object. Through this structured process, the environment attains a state suitable for dynamic testing and downstream real-time simulation.

Conversion of the annotated data to a JSON representation will happen when the user saves their projects or selects to export it.

4 Scenario Generator

4.1 Purpose and functionality

The simulation scenario is the cornerstone of the accessibility simulation. Through the scenario generator, users can describe simulation scenarios and execute them to get the accessibility assessment report. The simulation editor requires a Simulation Environment and a VUM as described in D5.1. In the application, users can create simulation scenarios through the user interface and describe the steps that the simulation should execute. The combination of these three inputs, namely 3D Simulation Environment, VUM and Simulation Scenario will output a detailed accessibility report regarding the provided inputs. Note that the 3D Environment and the Simulation Scenario are tangled together, thus the usage of the same simulation scenario is not possible in another environment. Contrary, the VUM is not restricting, and users can execute the same environment-scenario tuple with multiple VUMs to test for different kinematic, audio or visual challenged individuals.

4.2 Simulation preparation

This Section describes the steps that the user should follow to successfully execute an accessibility simulation. The core steps are the environment loading, the simulation scenario creation, and the virtual user model selection.

4.2.1 Environment loading and initialization

The first step for the simulation is to load a correct simulation environment. The simulation accepts only files from the simulation creation application (Section 3). Through the user interface, the user can import the desirable file.

Upon the finish of the loading process, the user will be prompt with a message regarding the completion status. In case of an error, the user will see an error message with a code and a brief description. In case of a successful load, the user will see the simulation environment in the 3D viewport. The user can also see all the interactable objects that had been configured through the 3D Environment Generator application.

4.2.2 Simulation scenario creation

The simulation scenario creation is the step where the user configures the steps that the simulation will execute. The user interface provides buttons to add, configure and remove simulation steps. In this step the user must carefully decide which simulation steps to add to the scenario. We suggest testing small and meaningful scenarios in order to keep the simulation as light as possible. The size of the simulation is not going to impact the output, but it will significantly slow the simulation. The scenarios can also be saved for later usage, but keep in mind that each scenario will be executable only in the same simulation environment, as they are closely tangled.

4.2.3 Virtual User Model selection

The last step is to select the virtual user model for the simulation. Users are able to load the VUM through a list with all available VUMs. The model can be changed before each new simulation in order to test the same environment with different user models. After selecting the VUM, the application will go under a load process which will configure the simulation's Avatar based on the selected VUM. The simulation avatar is a detailed rig holding information regarding all kinematic, auditory and visual

challenges of the selected model. The set-up process is responsible to configure control points and effectors for each bone and joint on the Avatar. Once the setup is completed users are able to execute the simulation scenario.

4.2.4 Simulation execution

This section presents a detailed technical description of the simulation generator. For each subsystem, the rationale behind the final implementation decisions is discussed, followed by a technical breakdown of the components developed for the application.

The Unity game engine was chosen for this accessibility simulation application because it provides an effective framework for integrating procedural animation, real-time simulation, and dynamic analysis within a unified environment. During the initial phase we tested multiple approaches on achieving accurate results namely, PID controlled joints, articulated bodies and Procedural animation kinematics with inverse dynamics analysis. After the initial testing we concluded that the latter option had the most reliable output. This approach uses a combination of procedural animation to create a kinematic plan and followed by an inverse dynamics algorithm which calculates the forces and torques acting on the avatar during the animation playback. This approach provides a flexible bases that will allow us to adapt the simulations in multiple use cases while providing accurate results.

4.3 Kinematic challenges simulations

4.3.1 Kinematic planning

The kinematic planning is the first subsystem which calculates the kinematic plan for the avatar. The planning consists of four steps, the simulation step breakdown, the path finding, the gait generation and the frame generation.

4.3.1.1 Simulation step breakdown

The simulation step breakdown, executes an algorithm that reads the provided simulation scenario and breaks it down to its basic steps. Each scenario step (waypoint, interaction) executes a different algorithm and outputs a different configuration that will be passed down to other components of the kinematic planning subsystem. A type-dependency diagram is shown in Figure 11.

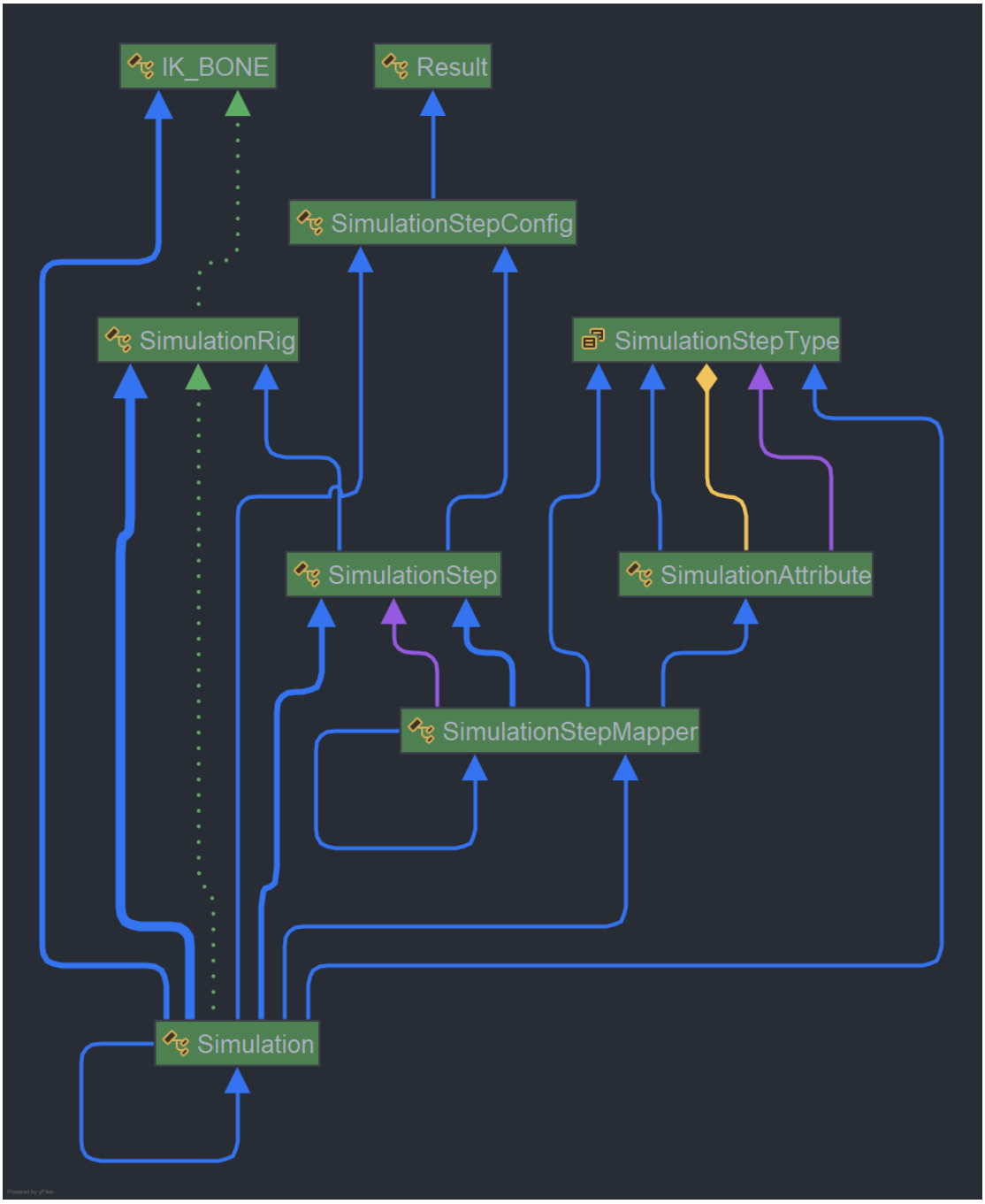

Figure 11. Simulation step breakdown type dependency diagram

The architecture allows us to further expand the set of algorithms for the simulation step breakdown. The algorithm is split in two parts, the first one determines which type of simulation step should be used, thus allowing to pick different types of algorithms based on the scenario's VUM. The second part is the algorithm that will later generate the primitive animations (Gaits). Each algorithm is responsible to determine which gaits are necessary to most efficiently execute the scenario step. The output of this subsystem is a list of the necessary gaits for the simulation. In case of an error in this step, an early simulation error will be thrown. This will result in a really abstract assessment, and mainly means that the provided configuration (Environment, VUM and scenario) is not executable.

4.3.1.2 Gait creation

The gait generation is initiated after the successful simulation step breakdown. Each gait is responsible to create a trajectory for the bones/joints or the inverse kinematic points it controls. The selected architecture completely decouples the gaits from the simulation step breakdown in order to allow us to optimize, change and add gaits in the future. A type-dependency diagram is show in Figure 12.

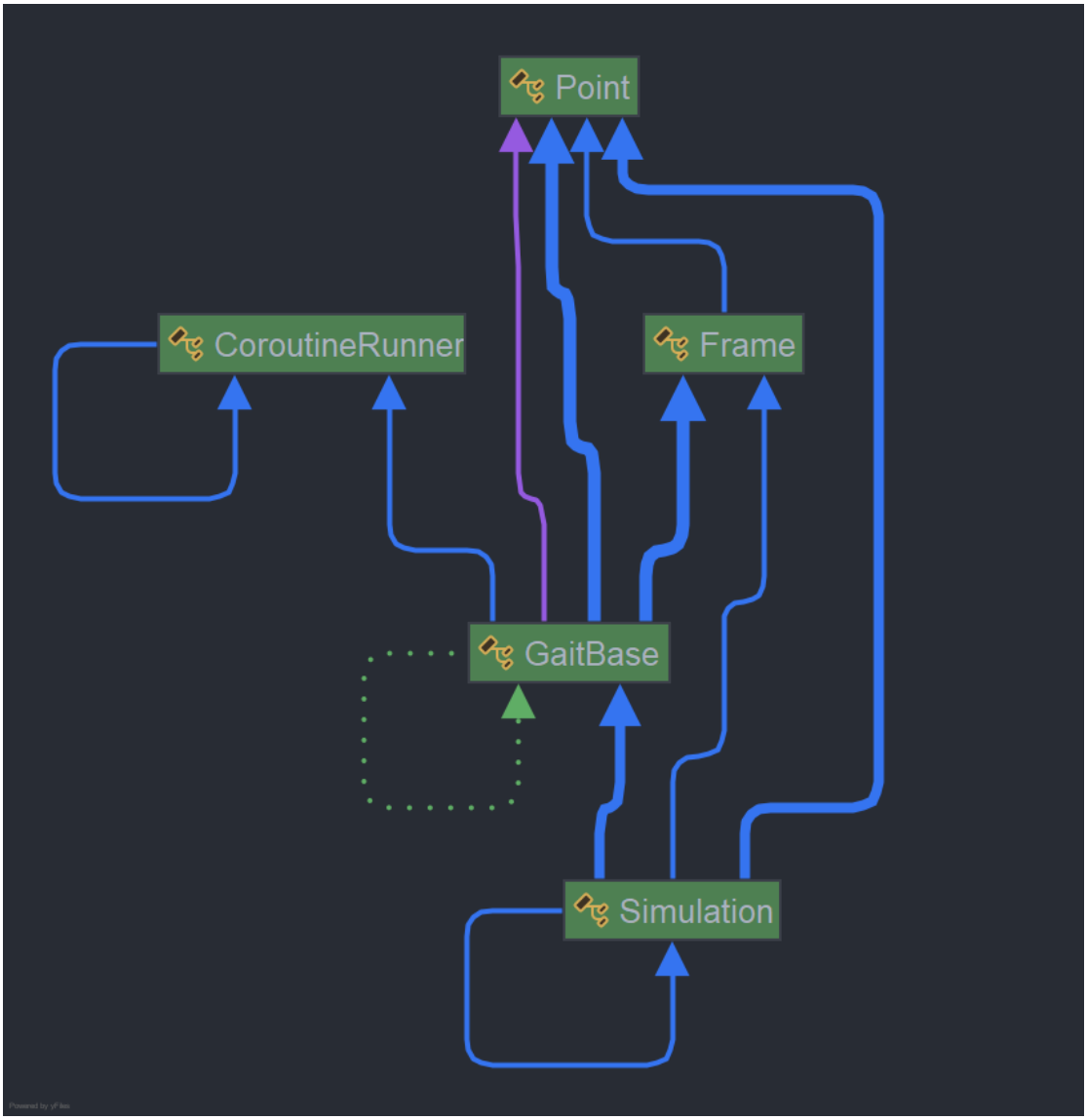

Figure 12. Gait generation type dependency diagram

For each gait, an algorithm which generates the trajectories of bones involved in the primitive task/movement is implemented. Depending on the movement, the trajectory might describe the rotation of each joint or the trajectory of an inverse kinematic target. The gait has two phases, a plan and an execute phase. The execute phase is common for all gaits and implemented on the abstract GaitBase class while the plan phase executes the algorithm that each concrete gait implements. The algorithm must calculate all the trajectories and store them in a dictionary for later use.

The core gaits that have been developed for the first version are

1. Reach: Moves the inverse kinematics target of the hand in order to reach a provided target;
2. Step: Moves the inverse kinematics target of the foot in order to complete a step;
3. Grab: Rotates the fingers in order to grab an object.

With a combination of these gaits, the basic scenarios can be described and tested.

In order to provide a common bases for all gaits, the store is done in “Frames”. Each frame contains information about all the controlled targets, their trajectory points that are sampled automatically and a timestamp that indicates the start time of the frame.

4.3.2 Dynamic simulation

The dynamic simulation executes the aforementioned kinematic plan using the generated frames from the gaits. During the execution, an inverse dynamics algorithm calculates the forces that act on bones and joints during the movement. These calculations are used from the assessment module in order to provide meaningful feedback to the user.

4.3.2.1 Inverse Dynamics

The inverse dynamics module computes the internal joint forces and torques required to produce the observed motion of the Avatar during the execution of the animation generated from the inverse kinematic planning subsystem. It implements a Newton–Euler recursive algorithm that operates entirely in world space to ensure physical consistency across complex kinematic chains. During each simulation step, the module first performs a **forward recursion**, propagating angular velocity, angular acceleration, and linear acceleration from the root segment through all connected links, using each segment’s local motion data derived from Unity’s transforms. It then executes a **backward recursion**, starting from the terminal segments and moving upward, where it calculates the resultant forces and torques on each segment based on its mass, inertia, and external loads such as gravity or contact forces. This approach enables real-time estimation of physically consistent joint reactions and muscle or actuator loads.

It is important to note that in order for the inverse dynamics to calculate the correct forces, any external forces, not inherently from the movement, such as ground reaction forces, must be added manually following an algorithmic logic. In order to achieve that, another component was created which acted as a collision identifier, named external force manager. This component is responsible to identify collision points between the avatar and the world and apply the correct reaction forces to the correct segment.

The overall inverse dynamics subsystem is shown in Figure 13.

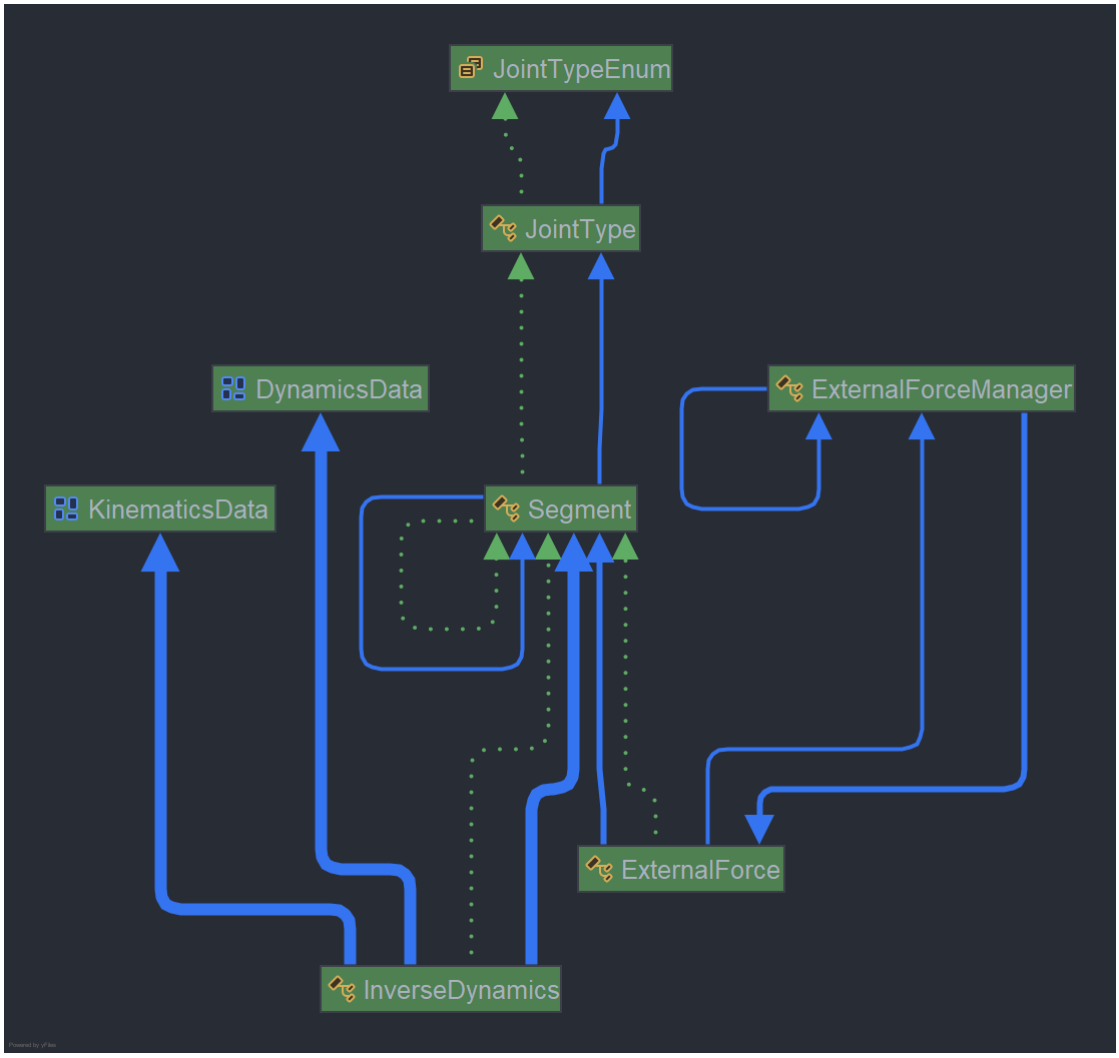

Figure 13. Inverse dynamics type dependency diagram

4.4 Hearing challenges simulations

The simulation of hearing impairments was approached through a combination of Unity's AudioMixer system, spatial audio features, and carefully curated sound design. Each condition was interpreted from a physiological and perceptual standpoint, then implemented using real-time audio processing techniques, tailored to perform efficiently within the Meta Quest 3 mobile XR environment. To manage these conditions, we developed a centralized script that dynamically switches audio states through AudioMixer snapshots and parameter blending. This allows seamless transitions between auditory profiles during XR experiences. Below, we describe each impairment and the technical solution implemented.

4.4.1 High-Frequency Hearing Loss

This condition reflects the inability to perceive high-pitched sounds—often seen in age-related hearing loss or noise-induced damage to the cochlear hair cells. To simulate this, we implemented a low-pass audio filter within Unity's AudioMixer. The filter attenuates frequencies above 2,000–3,000 Hz, removing sounds such as alarms, chirps, or high-pitched speech cues. This effect is applied globally to all audio output, creating an experience where the soundscape feels dull or incomplete. The filter can

be enabled or adjusted dynamically via script, offering an effective approximation of this common auditory deficit.

4.4.2 Unilateral Hearing Loss

Unilateral or one-sided hearing loss disrupts binaural sound perception, affecting a person's ability to localize sound or detect its direction. In our simulation, we mimicked this effect by muting or significantly reducing audio output on one stereo channel, either left or right. Using Unity's Stereo Pan control or AudioMixer parameters, we created the sensation of hearing only through one ear, which is especially impactful in VR, where spatial sound plays a crucial role. This implementation helps users understand how environmental awareness is compromised when binaural hearing is lost.

4.4.3 Tinnitus

Tinnitus is characterized by a persistent internal ringing, buzzing, or high-pitched sound that is not caused by any external source. To simulate this condition, we created a looping audio source containing a high-frequency tone and attached it to the camera (i.e., the player's head), so that it always remains present in the user's perception. The pitch, stereo positioning (left/right/both ears), and volume of the ringing can be adjusted to match different forms of tinnitus. This subtle but constant sound serves as a distraction and demonstrates how intrusive tinnitus can be, especially in moments of silence.

4.4.4 Muffled Hearing (Conductive Hearing Loss)

Conductive hearing loss results in sounds being perceived as muffled, distant, or as if heard through a barrier, often due to fluid buildup, infections, or physical blockage. We simulated this by applying a bandpass filter that preserves only the mid-range frequencies (around 500–2,000 Hz) while attenuating low and high frequencies. This effect was combined with mild reverb and slight volume reduction, creating an acoustic profile similar to hearing underwater or through closed doors. The result gives users the impression of degraded sound clarity and loss of detail, which can affect speech understanding and environmental awareness.

These hearing impairments were designed to be switchable at runtime, either through in-VR interaction, scripted sequences, or a UI dashboard. Each one was modeled based on real-world clinical descriptions and validated through informal testing.

4.5 Visual challenges simulations

In this chapter, we are going to explore how we simulated the visual impairments in the project. The objective of this task is to implement a suite of shader-based visual simulations representing common visual impairments, allowing users to experience the visual world as perceived by individuals with various vision-related disabilities. These effects form a core component of our accessibility awareness module, to be tested in VR on the Meta Quest 3.
The development was implemented in Unity, initially using the Built-in Render Pipeline with OnRenderImage() post-processing, and later adapted to the Universal Render Pipeline (URP) to support Meta Quest 3 deployment.
A unified C# controller script (VisualImpairmentEffect.cs) was created to manage and dynamically switch between different visual effect modes. Each mode is rendered via a custom full-screen shader, simulating a specific visual impairment. The system is modular and optimized for mobile performance.

The visual impairments simulated are:

- Color blindness:
  - Protanopia – Red color blindness
  - Deuteranopia – Green color blindness
  - Tritanopia – Blue color blindness
- Myopia;
- Macular degeneration.

For colour blindness, each subcategory was simulated using a colour transformation matrix applied in a shader. The matrix remaps RGB values based on the physiology of cone cells affected in each condition. This is implemented as a pass-through full-screen shader in URP using a matrix multiply per-pixel. The core idea is to remap RGB colours to simulate the colour perception of individuals lacking one type of cone cell in the retina. Each form of colour blindness (red-, green-, or blue-deficient) corresponds to a specific 3x3 colour matrix derived from colour vision research.

These matrices simulate how a person with the condition would perceive standard colours by altering pixel values in real time. The shader samples the original scene colour and multiplies it by the selected matrix, producing a filtered result. Figures 14, 15 and 16 show the user view with each of the three-color blindness subcategories (protanopia, deuteranopia, tritanopia).

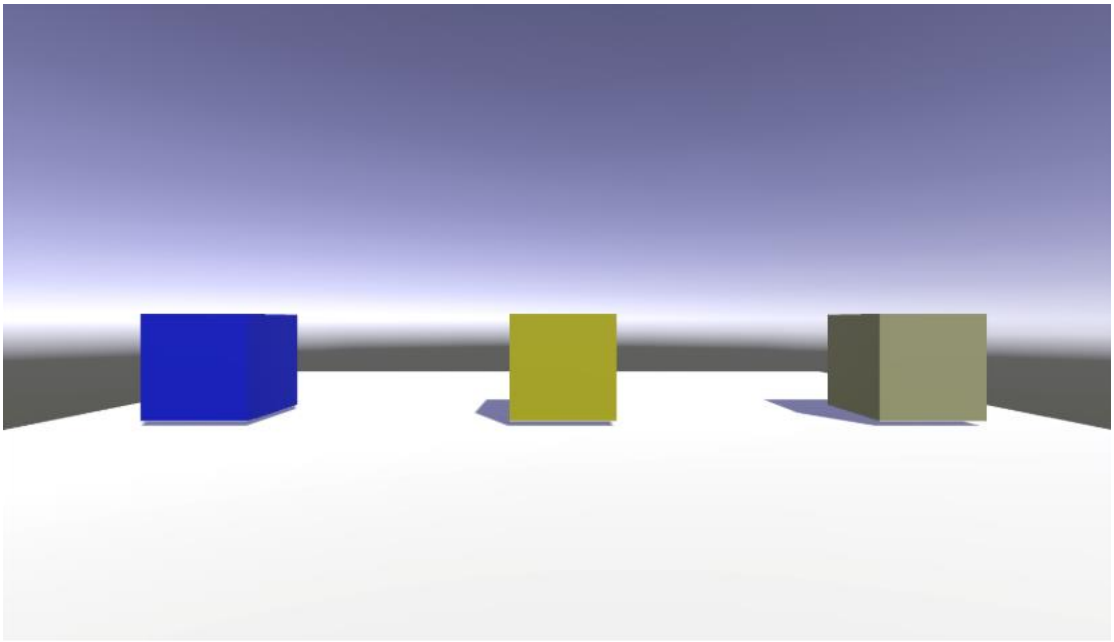

Figure 14. Protanopia example

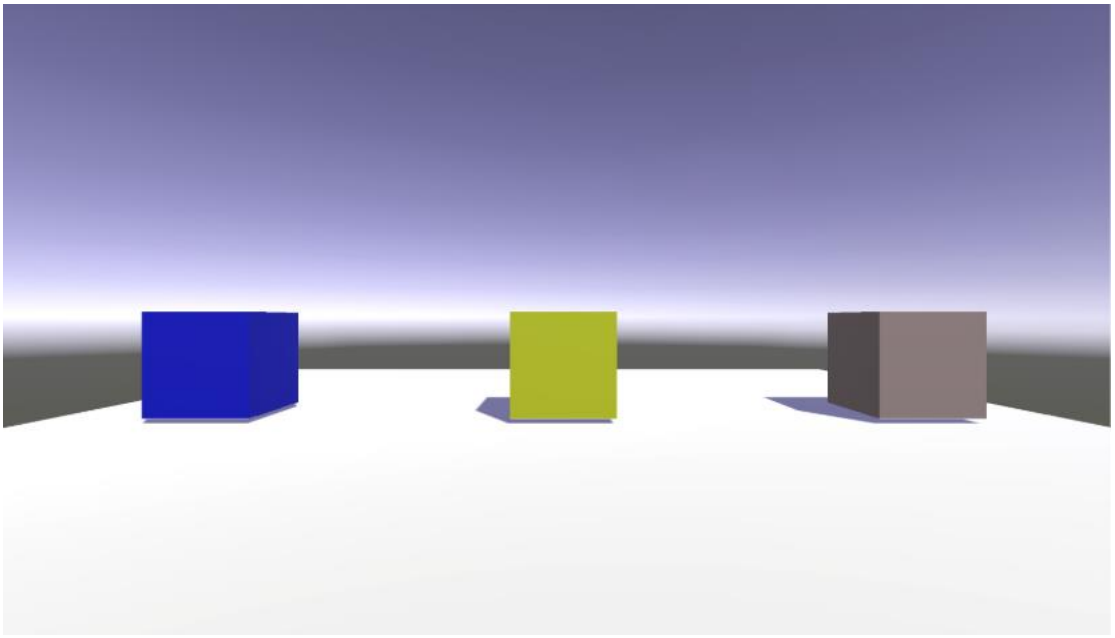

Figure 15. Deuteranopia example

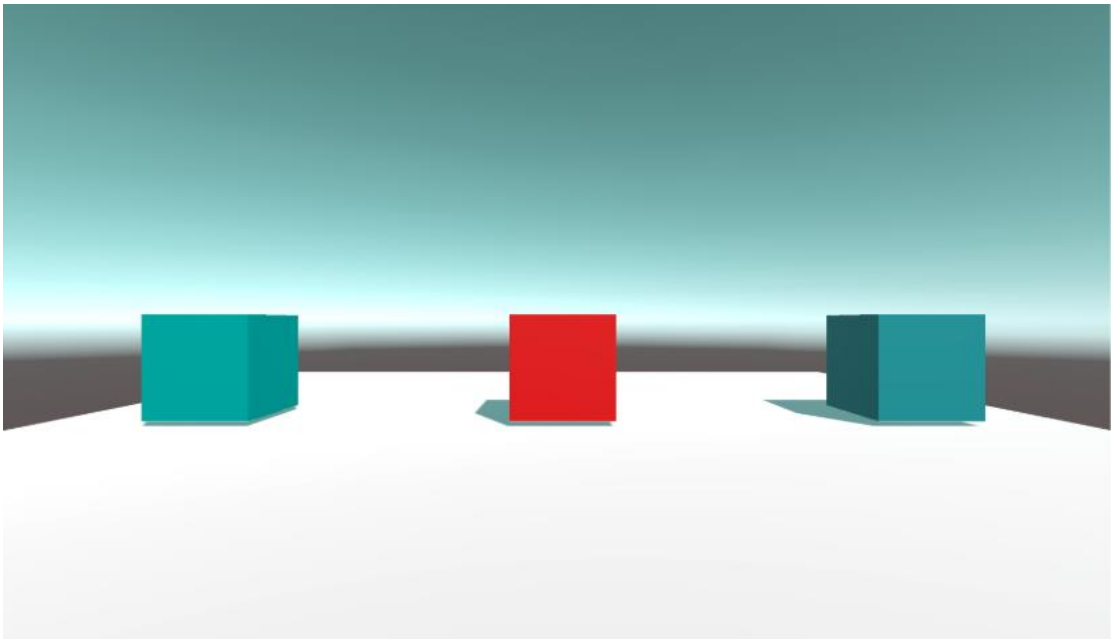

Figure 16. Tritanopia example

For myopia, a depth-based blur shader was developed. Using Unity's _CameraDepthTexture, each pixel is blurred proportionally to its distance from the camera. The effect simulates the inability of the eye to focus on distant objects. To simulate nearsightedness, the shader measures the distance of each pixel from the camera using a linearized depth value. A blur effect is applied based on this depth: closer objects stay sharp, while distant objects become blurred. The blur itself is implemented as a box blur kernel, sampling a small region around the pixel. The blur intensity is modulated by the depth, giving a natural falloff effect. Figure 17 shows the myopia effect on the cubes with different distances between from the user.

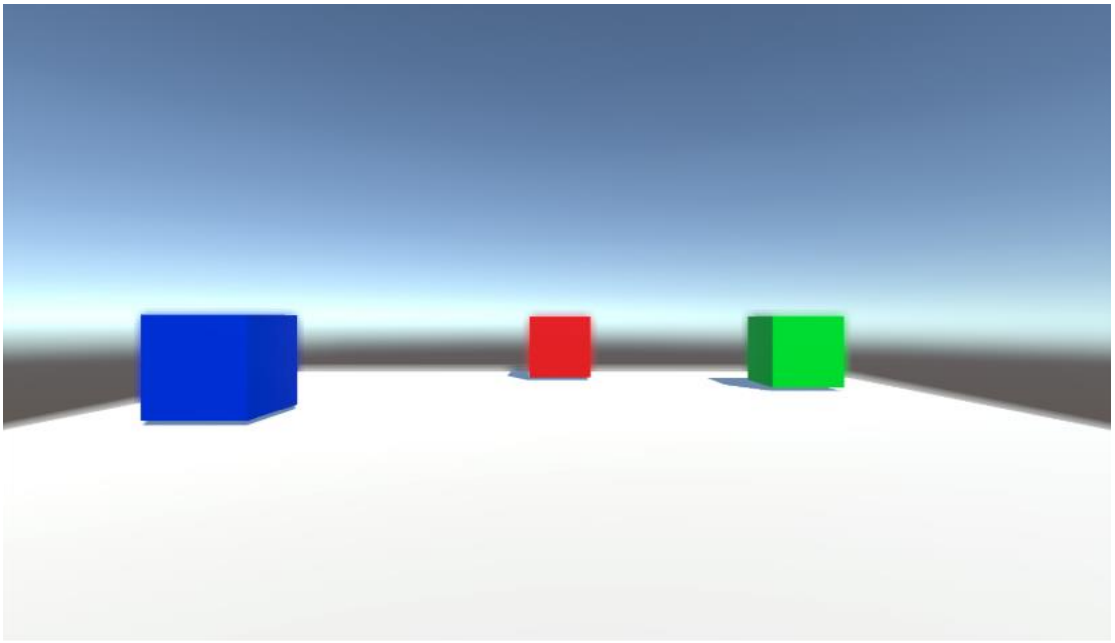

Figure 17. Myopia example

For macular degeneration, a circular screen-space mask centered on the camera simulates central blur and dimming. The blurred region uses a soft transition ("feather") into normal vision. An optional black spot (scotoma) can be blended into the center to simulate advanced disease. Macular degeneration primarily affects central vision. The shader simulates this by creating a circular region in screen space (centered at the viewport center) where visual degradation occurs. The inner area is blurred and dimmed. A smoothstep transition (feathering) creates a soft boundary between affected and unaffected regions. An additional multiplier allows simulating scotomas—dark or grey spots in the center. Figure 18 shows how a patient with macular degeneration sees.

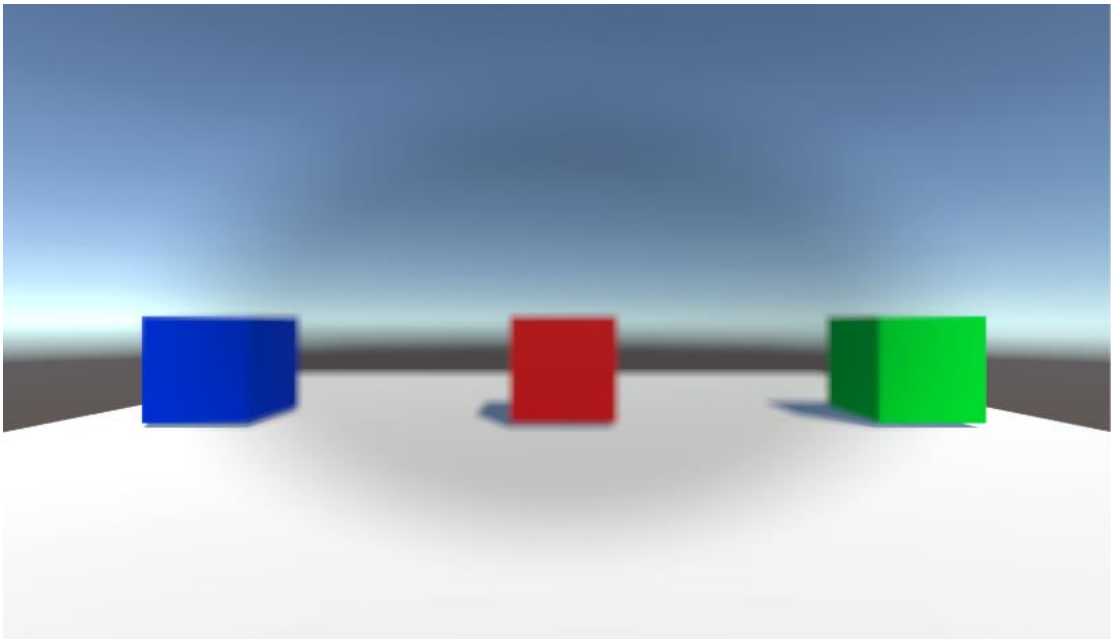

Figure 18. Macular degeneration example

Finally, we have created a script for controlling and changing visual impairments based on user preference. The VisualImpairmentEffect script serves as a runtime shader controller, attached to the camera. It dynamically selects and applies the appropriate fullscreen visual effect (shader) based on

user input or a simulation scenario. The script supports switching between multiple impairments such as color blindness, myopia, and macular degeneration. Originally based on OnRenderImage() (for Built-In Pipeline), it was later modularized to support URP-compatible renderer features and mobile VR deployment on Meta Quest 3.

Some key functionalities include:

- **Effect Mode Enum:** An internal enum (VisualEffectMode) defines available simulation models: Normal, Protanopia, Deuteranopia, Tritanopia, Myopia, MacularDegeneration;
- **Shader Management:** The script holds references to each shader (Shader) and corresponding runtime Material. On initialization, it creates Material instances and ensures the camera provides a depth texture (if needed, for myopia);
- **Runtime Switching:** Based on the selected mode, the script applies the appropriate Material using Graphics.Blit() in OnRenderImage(). It also passes relevant parameters (e.g., blur strength, dimming) into the active shader.

Parameters like blur radius and scotoma strength are tunable from the Inspector, allowing performance balancing on mobile XR devices like the Meta Quest 3.

4.6 Assessment

The assessment subsystem collects data from the dynamic simulation and executes an algorithm that outputs a meaningful report to the user. Using the calculated forces acting on each joint and comparing it to the provided VUM's information, critical errors are identified. These errors encapsulate the joint and the position information which is cached from the simulation. In addition, the assessment subsystem is able to provide warnings regarding sustain loads that are within the VUM's range, but could introduce fatigue. The report is exportable and can be saved for later use.

4.7 User interface overview

For the first version of the scenario generator, our objective was to identify the fundamental user interface components required for user interaction, prior to focusing on user experience and quality-of-life enhancements. Accordingly, the first mockup includes the following elements:

- A simulation creation menu that allows users to add, configure or remove simulation steps (Figure 19);
- Two buttons allowing the user to load VUMs and environments exported from the simulation environment application (Figure 19);
- A simulation menu that presents real-time information during the execution of the simulation (Figure 20);
- A button to initiate the simulation process (Figure 20);
- A results menu that displays the simulation output along with a brief assessment (Figure 21).

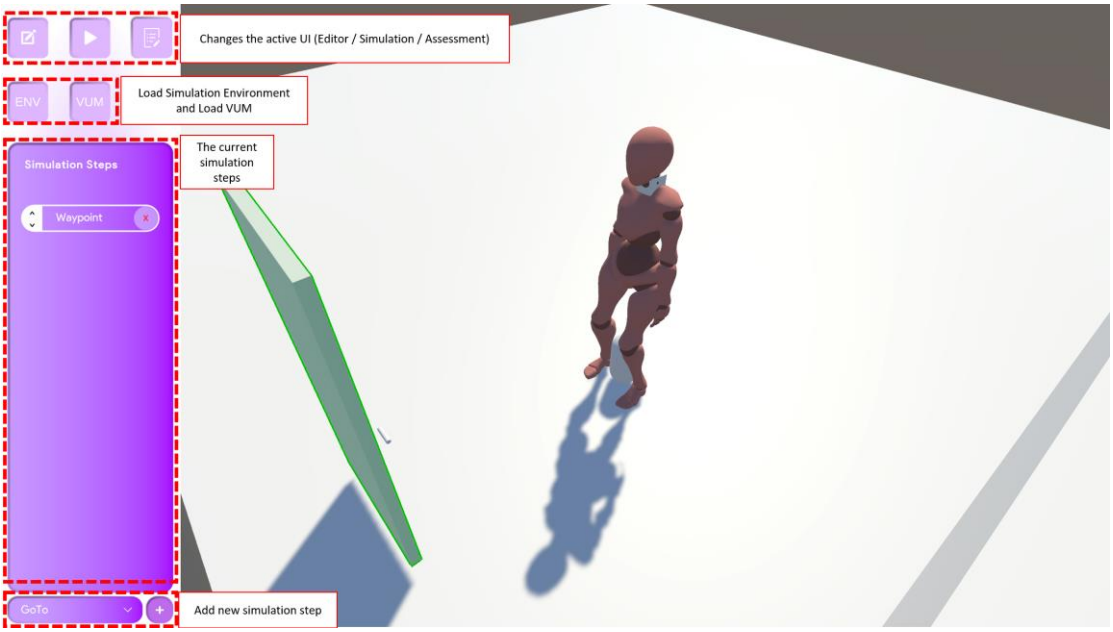

Figure 19. Simulation creation UI

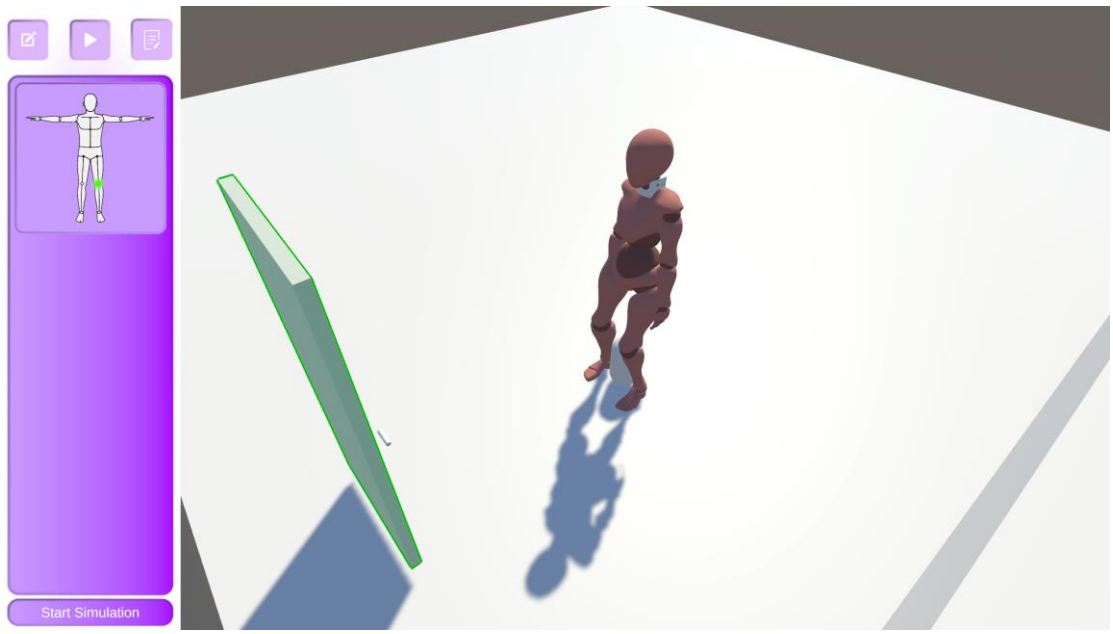

Figure 20. Simulation execution UI

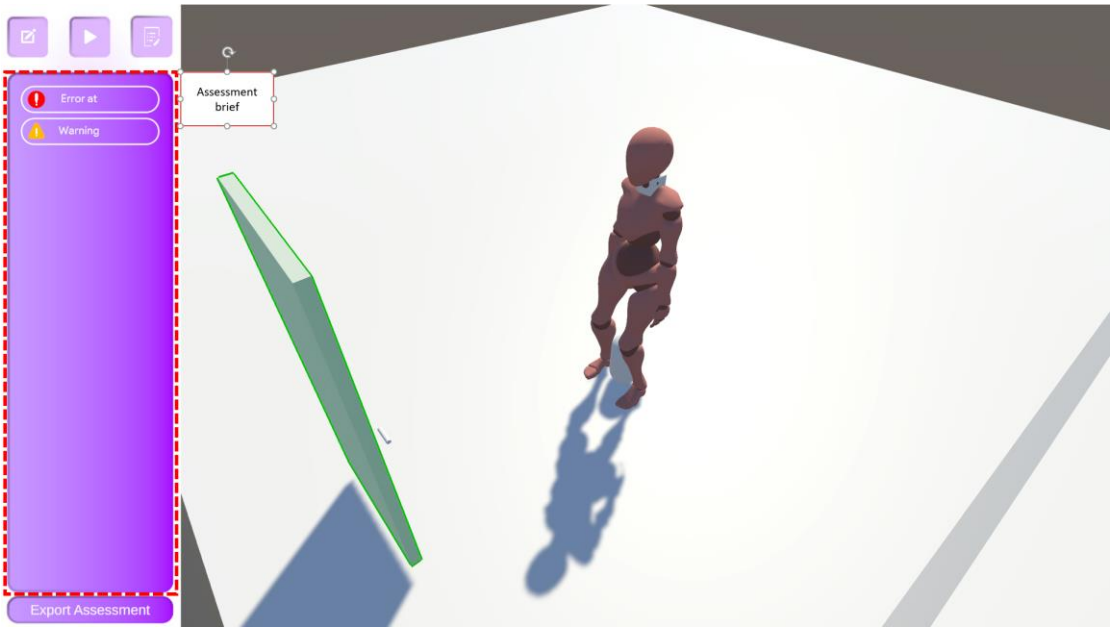

Figure 21. Simulation assessment UI

5 Testing and validation

5.1 Unit and integration testing

To enable dynamic personalization and testing of impairment simulations, we integrated an external API provided by the VUMS platform. This API exposes endpoints that deliver structured data about users, including their associated impairments, demographic data, and simulation profiles. The API is accessed through a secure connection using a Bearer Authentication Token, managed within Unity using UnityWebRequest. The base URL for the API is defined in the Postman environment (https://euproject-access-forth.eu), and endpoints such as GET /api/v1/people and GET /api/v1/impairments allow Unity to fetch real-world-inspired data models. These models describe the specific conditions a virtual user has, for example, whether they experience protanopia, tinnitus, or myopia. In Unity, this data is retrieved at runtime and parsed from JSON into structured C# objects (UserProfile, ImpairmentProfile, etc.). Based on this information, the application dynamically activates the appropriate shaders or audio filters using the systems previously described. For instance, if a person retrieved from the API has "impairment": "MacularDegeneration", the corresponding visual shader is activated in the camera pipeline. This integration allows for scenario-based testing. Developers, researchers, or educators can predefine virtual user profiles in the backend, then experience the environment from their perspective in VR. Furthermore, this system can be extended to allow switching between users within the app interface, making the simulation highly adaptable and personalized.
The Unity implementation includes:

- A secure authentication process using the API token;
- Dynamic UnityWebRequest logic to call endpoints and parse JSON;
- Impairment Manager scripts that link retrieved impairments to runtime shader/audio activations.

This modular structure supports future enhancements, such as loading severity levels (e.g., mild or advanced macular degeneration), linking hearing and vision combinations, or integrating additional APIs related to accessibility or health simulations.

5.2 Simulation accuracy verification

The accuracy of the simulation is a critical issue and during the first version we aimed to verify that the output of each subsystem is optimal. In order to measure the accuracy of the simulation, we conducted a literature review to find information regarding joint force measurements during simple tasks that we would be able to reproduce in the simulation to validate the results. The literature provided us with plenty of insights regarding forces acting on the knees during walking cycles. By comparing the results of different studies, we found out that the peak forces were on the heel strike and the toe off stances, and the forces were ranging from 1-3.5 times the subject's bodyweight, depending on the walking speed and individual style. A reference of a full walking cycle can be seen in Figure 22, while a reference knee load from literature is presented in Figure 23.

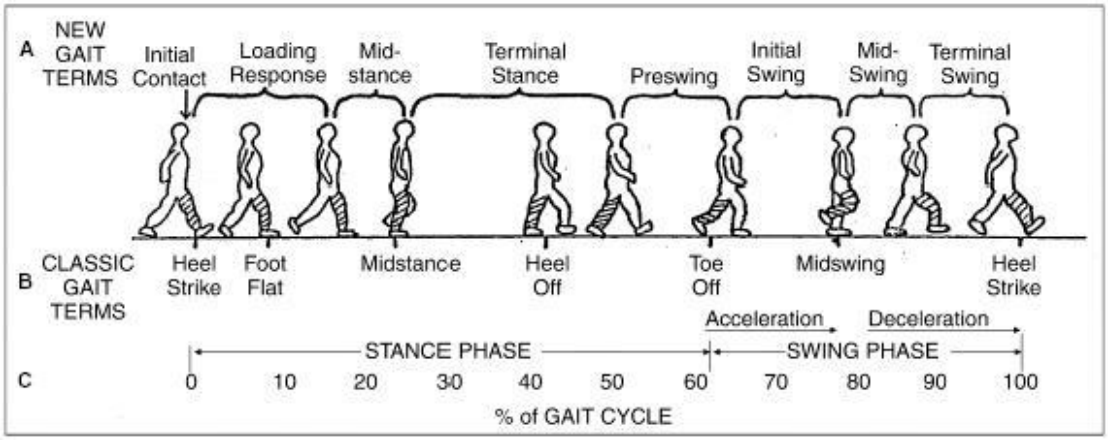

Figure 22. Reference of a walking cycle

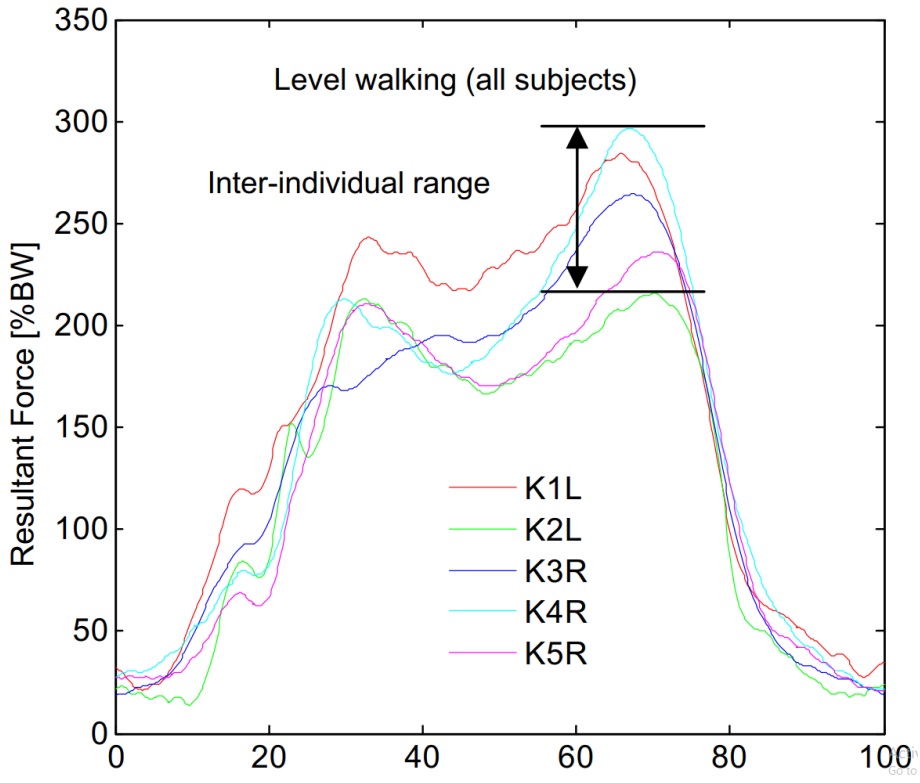

Figure 23. Walking cycle knee loads across 5 subjects [48]

To properly evaluate the simulation engine, we chose three scenarios:

1. Normal walking;
2. Ascending stairs;
3. Descending stairs.

The validation scenarios had only one simulation step with the underlying task of reaching a waypoint. As mentioned in section 4.2 there are three algorithmic steps that the simulation engine executes to generate the kinematic plan, the simulation step breakdown, the gait generation and the frame sampling. The kinematic plan output of the testing scenarios is shown in Figure 24.

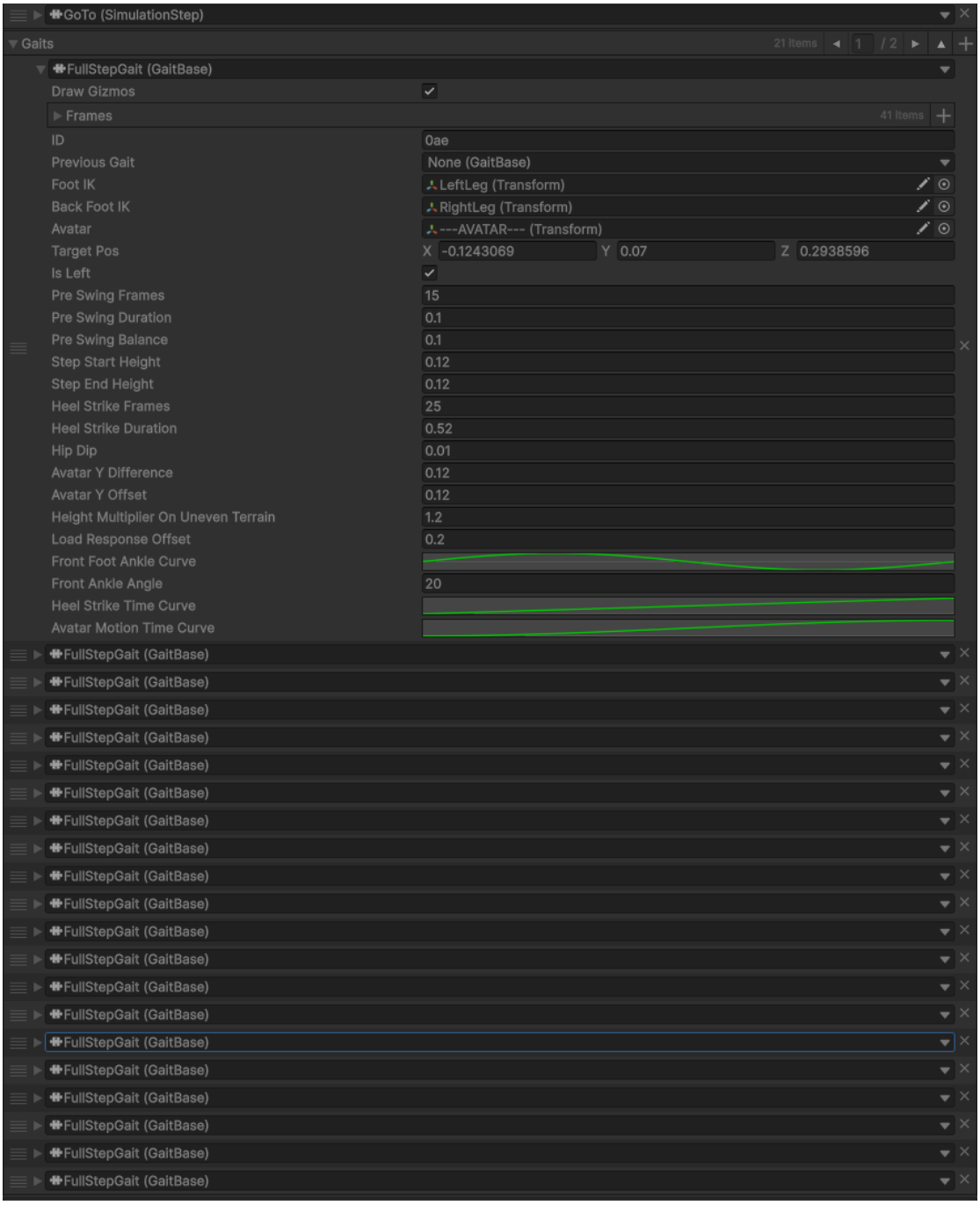

Figure 24. Testing scenario generated gaits as shown in Unity's Inspector

The dynamic simulation results aligned well with published data for normal walking. However, in both the ascending and descending stair scenarios, the knee joint forces showed unusually large peaks. In

this first version, we identified two main causes. In both stair scenarios, the inverse kinematics from Unity's Animation Rigging package introduced noticeable jitter. In the descending staircase, where the peaks were even more pronounced, we also discovered that the FullStepGait algorithm failed to adjust the step properly for the height change. In real movement, the toes make contact with the lower step before the heel, spreading the impact. Because this toe-first behavior wasn't represented in the algorithm, the model loaded the full weight directly onto the heel, creating a large, unrealistic spike in force (See Figure 25). Both issues are going to be fixed in the final version of the simulation editor, as part of the integration activities and testing (T7.2, T7.3).

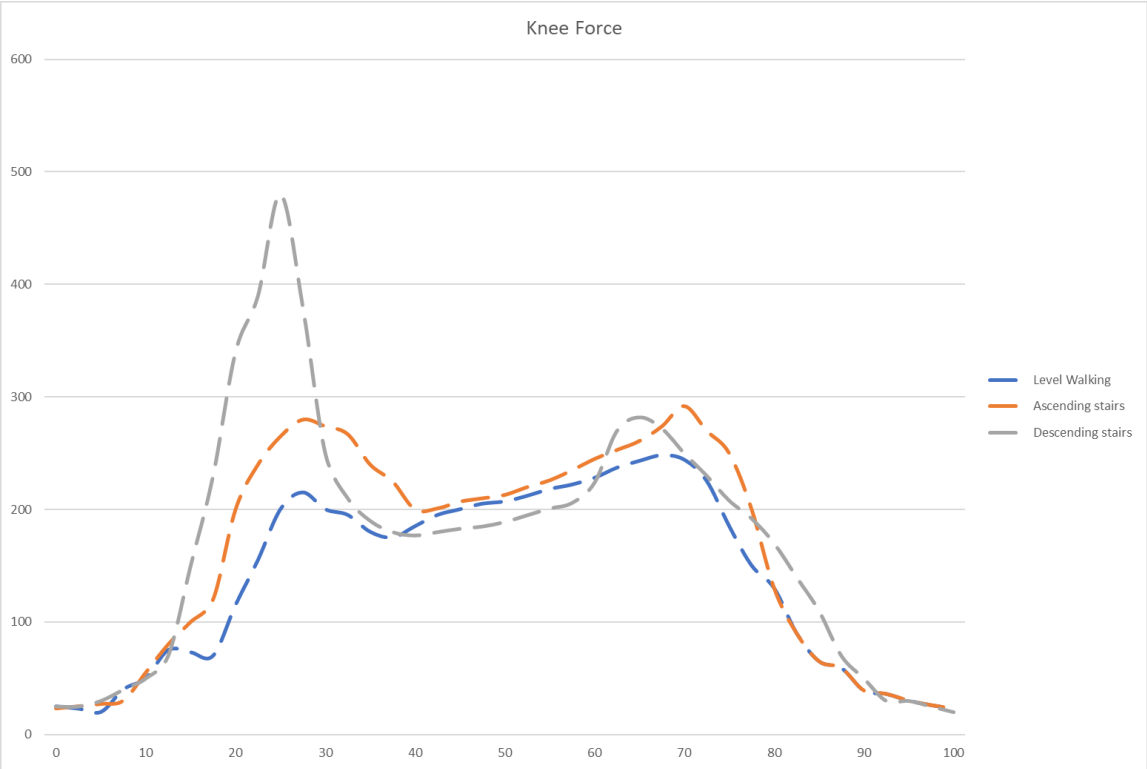

Figure 25. Forces calculated from the inverse dynamic algorithm during the test scenario

6 Conclusions

This first version of the 3D Simulation Environments establishes the technical groundwork needed for the AccesS accessibility simulation platform. The work delivered here demonstrates that 3D building models can be transformed into structured, semantically annotated, and interaction-ready environments suitable for downstream simulation workflows. Through the SANTO tool, static IFC elements are enriched with behavioural metadata, kinematic affordances, and standardized interaction schemas, enabling accurate and reproducible simulations across a wide range of architectural scenarios.

On top of these environments, the Scenario Generator provides the mechanisms for defining simulation sequences, selecting Virtual User Models, and executing multi-stage simulations that combine kinematic planning, procedural animation, inverse dynamics, and sensory impairment modelling. The integration of these subsystems confirms that the platform can produce meaningful accessibility assessments grounded in measurable forces, trajectories, and perceptual constraints.

The testing and validation activities carried out in this version demonstrate promising alignment with published biomechanical data for simple walking tasks. They also highlight areas requiring refinement, particularly the handling of stair navigation, IK stability, and contact modelling. These insights guide the improvements planned for the next iteration of the system, where enhanced gait algorithms, improved environmental interactions, and expanded validation datasets will further strengthen accuracy and reliability.

7 References

[1] Zaman, Ahmad Akib Uz, Ahmed Abdelaty, and Md Habibur Rahman Sobuz. "Integration of BIM data and real-time game engine applications: Case studies in construction safety management." J. Inf. Technol. Constr. 29 (2024): 117-140.

[2] Wu, Zhong-Qi, and I-Chen Wu. "Design and Implementation of an IFC Data Model to Unity Data Model Transformation Mechanism." In International Conference on Computing in Civil and Building Engineering, pp. 1609-1614. 2016.

[3] Industry Foundation Classes. Wikipedia, The Free Encyclopedia. Available online: https://en.wikipedia.org/wiki/Industry_Foundation_Classes (accessed on 11 November 2025).

[4] Zaman, Ahmad Akib Uz, Ahmed Abdelaty, and Md Habibur Rahman Sobuz. "Integration of BIM data and real-time game engine applications: Case studies in construction safety management." J. Inf. Technol. Constr. 29 (2024): 117-140.

[5] Nandavar, Anirudh, Frank Petzold, D. Nassif, Gerhard Schubert, and B. Ag. "Interactive virtual reality tool for BIM based on IFC." In Learning, Adapting and Prototyping, Proceedings of the 23rd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), pp. 453-462. Tsinghua University, Beijing, China: Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) in Hong Kong, 2018.

[6] Autodesk, Inc. 2013. "Door Types." In Autodesk Revit 2013 Help Documentation. Autodesk Help. Available online: https://help.autodesk.com/view/BDS/2013/ENU/caas.html?url=caas/vhelp/help-dev-autodesk-com/v/Revit/enu/2013/Help/00001-Revit-He0/0328-Build-th328/0330-Architec330/0389-Doors389/0397-Door-Typ397.html (accessed on 11 Nov 2025).

[7] Gourlis, Georgios, and Iva Kovacic. "A holistic digital twin simulation framework for industrial facilities: BIM-based data acquisition for building energy modeling." Frontiers in Built Environment 8 (2022): 918821.

[8] Jeong, David C., Steffie Sofia Yeonjoo Kim, Jackie Jingyi Xu, and Lynn C. Miller. "Protean kinematics: A blended model of VR physics." Frontiers in Psychology 12 (2021): 705170.

[9] NVIDIA Corporation. 2025. "What Is SimReady?" In NVIDIA Glossary. Available online: https://www.nvidia.com/en-us/glossary/simready/ (accessed on 11 Nov 2025).

[10] Unity Technologies. 2023. "Inverse Kinematics." In Unity Manual. Available online: https://docs.unity3d.com/6000.2/Documentation/Manual/InverseKinematics.html (accessed 11 Nov 2025).

[11] Darmstadt Graphics Group GmbH (DGG). 2025. "The Complete glTF to Revit Conversion Guide." RapidPipeline. Available online: https://rapidpipeline.com/en/a/conversions-gltf-to-revit/ (accessed on 11 Nov 2025).

[12] Autodesk, Inc. 2026. "3ds Max 2026 Help." In Autodesk 3ds Max 2026 Documentation. Available online: https://help.autodesk.com/view/3DSMAX/2026/ENU/?guid=GUID-5B4C8EC2-2230-4F9F-B3C6-48D9E347E37D (accessed on 11 Nov 2025).

[13] Esri. 2025. "Import glTF 3D Models | Sample Code." In ArcGIS Maps SDK for JavaScript – Sample Code. Available online: https://developers.arcgis.com/javascript/latest/sample-code/import-gltf/ (accessed on 11 Nov 2025).

[14] Bimdots. 2025. "glTF Exporter." In Bimdots Manuals. Available online:

https://bimdots.com/manuals/gltf-out/ (accessed on 11 Nov 2025)

[15] XReco. 2025. “JSON Scene Format for 3D Assets and Cloud.” Available online: https://xreco.eu/json-scene-format-for-3d-assets-and-cloud/ (accessed on 11 Nov 2025).

[16] Library of Congress. 2025. “glTF (GL Transmission Format) Family.” In Sustainability of Digital Formats: Planning for Library of Congress Collections. Available online: https://www.loc.gov/preservation/digital/formats/fdd/fdd000498.shtml (accessed on 11 Nov 2025).

[17] Edwards, Gareth, Haijiang Li, and Bin Wang. "BIM based collaborative and interactive design process using computer game engine for general end-users." Visualization in Engineering 3, no. 1 (2015): 4.

[18] Think Design. 2025. “Contextual Menus: How Can They Enhance Usability.” In Think Design Blog. Available online: https://think.design/blog/contextual-menus-how-can-they-enhance-usability/ (accessed on 11 Nov 2025).

[19] Bricsys NV. 2025. “Structure panel configurations for BIM.” In BricsCAD BIM Help Documentation. Available online: https://help.bricsys.com/en-us/document/bricscad-bim/building-data/structure-panel-configurations-for-bim (accessed on 11 Nov 2025)

[20] PAAcademy. 2025. “Parametric Object Modeling in BIM.” Available online: https://paacademy.com/blog/parametric-object-modeling-in-bim (accessed on 11 Nov 2025).

[21] Pinnacle IIT. 2024. “Parametric Design in BIM.” Available online: https://pinnacleiit.com/blogs/bim/parametric-design-in-bim/ (accessed on 11 Nov 2025).

[22] Obayashi Corporation. 2025. “About hierarchical sorting methods.” In Smart BIM Standard (SBS). Available online: https://smartbimstandard.com/en/sbs/operation_method/ (accessed on 11 Nov 2025).

[23] Collao, Jorge, Fidel Lozano-Galant, José Antonio Lozano-Galant, and Jose Turmo. 2021. "BIM Visual Programming Tools Applications in Infrastructure Projects: A State-of-the-Art Review" Applied Sciences 11, no. 18: 8343. https://doi.org/10.3390/app11188343

[24] BIMCommunity. 2023. “Parametric Design with BIM Visual Programming.” In BIMCommunity. Available online: https://www.bimcommunity.com/parametric-design-with-bim-visual-programming/ (accessed on 11 Nov 2025).

[25] AlterSquare. 2025. “Open Source BIM: Complete Guide to Building on FreeCAD’s Foundation.” Available online: https://altersquare.io/open-source-bim-complete-guide-to-building-on-freecad-foundation/ (accessed on 11 Nov 2025).

[26] Jeong, WoonSeong, and Kee Han Kim. "A performance evaluation of the BIM-based object-oriented physical modeling technique for building thermal simulations: A comparative case study." Sustainability 8, no. 7 (2016): 648.

[27] Chen, Sou-Han, and Fan Xue. "Interactive BIM-based VR: A case study of doors." In International Conference on Computing in Civil and Building Engineering, pp. 245-255. Cham: Springer Nature Switzerland, 2022.

[28] International Alliance for Interoperability (IAI). The IFC2x3 Final Release Documentation: IfcDoorPanelProperties. Available online: https://iaiweb.lbl.gov/Resources/IFC_Releases/R2x3_final/ifcsharedbldgelements/lexical/ifcdoorpanelproperties.htm (accessed on 11 Nov 2025).

[29] Wu, Jin, and Jiansong Zhang. "Automated BIM object classification to support BIM interoperability." In Construction Research Congress 2018, pp. 706-715. 2018.

[30] Lin, Ya-Hong, Yu-Shen Liu, Ge Gao, Xiao-Guang Han, Cheng-Yuan Lai, and Ming Gu. "The IFC-based path planning for 3D indoor spaces." Advanced Engineering Informatics 27, no. 2 (2013): 189-205.

[31] Dassault Systèmes. 2025. "IFC Human." Available online: https://www.3ds.com/partners/solutions/SIT000039_PDT00000000961_IFC_Human (accessed on 11 Nov 2025).

[32] JSON Schema. 2025. "Specification." Available online: https://json-schema.org/specification (accessed on 11 Nov 2025).

[33] Al-Sadoon, Nidhal, and Raimar J. Scherer. "IFC semantic extension for dynamic fire safety evacuation simulation." In Proceedings of the CIB W78 Conference, Luxembourg, pp. 11-15. 2021.

[34] Noardo, Francesca, Ken Arroyo Ohori, Thomas Krijnen, and Jantien Stoter. 2021. "An Inspection of IFC Models from Practice" Applied Sciences 11, no. 5: 2232. https://doi.org/10.3390/app11052232

[35] Sorbi, Tommaso, Vito Getuli, Pietro Capone, and Farzad Pour Rahimian. "AGENT-BASED SIMULATION FRAMEWORK FOR ENHANCED CONSTRUCTION SITE RISK ESTIMATION AND SAFETY MANAGEMENT." Journal of Information Technology in Construction 29 (2024).

[36] Lee, Yong-Cheol, Charles M. Eastman, and Wawan Solihin. "Rules and validation processes for interoperable BIM data exchange." Journal of Computational Design and Engineering 8, no. 1 (2021): 97-114.

[37] 3D Design Bureau. 2023. "3D BIM Explained: The Foundation of a BIM Journey." Available online: https://3ddesignbureau.com/blog/3d-bim-explained-the-foundation-of-a-bim-journey/ (accessed on 11 Nov 2025).

[38] MDN Web Docs. 2025. "3D collision detection." In Techniques for game development. Available online: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection (accessed on 11 Nov 2025).

[39] 3D Collision Detection Tutorial - Collision Detection in JavaScript (YouTube video ID IG95gd5WRrg) (accessed on 11 Nov 2025).

[40] Park, Hyejin, David Stephen Panya, Hyungmo Goo, Teahoon Kim, and Jihyo Seo. "BIM-based virtual reality and human behavior simulation for safety design." In Proceedings of eCAADe 36th Annual Conference, vol. 2, pp. 823-832. 2018.

[41] The Khronos Group. 2021. "Enabling Consistent and Powerful Asset Product Metadata for 3D Commerce." Available online: https://www.khronos.org/blog/pervasive-asset-metadata-in-3dcommerce (accessed on 11 Nov 2025).

[42] Chen, Jui-Fa, Wei-Chuan Lin, and Li-Hao Yang. "Applying an Enhanced Path Finding Avatar for a Virtual Environment." Engineering Letters 15, no. 2 (2007).

[43] Isikdag, Umit, Sisi Zlatanova, and Jason Underwood. "A BIM-Oriented Model for supporting indoor navigation requirements." Computers, Environment and Urban Systems 41 (2013): 112-123.

[44] Venugopal, M., C. Eastman, R. Sacks, I. Panushev, and V. Aram. "Engineering semantics of model views for building information model exchanges using IFC." In Applications of IT in the AEC Industry: Proceedings of the 27th International Conference-CIB W, vol. 78, p. 2010. 2010.

[45] Du, Sang, Lei Hou, Guomin Zhang, Yongtao Tan, and Peng Mao. "BIM and IFC data readiness for AI integration in the construction industry: A review approach." Buildings 14, no. 10 (2024):

3305.

[46] Ma, Yu-Pin. "Improved interaction of BIM models for historic buildings with a game engine platform." Applied Sciences 12, no. 3 (2022): 945.

[47] Shneiderman, Ben. "Direct manipulation: A step beyond programming languages." Computer 16, no. 08 (1983): 57-69.

[48] Kutzner, I. *et al.* (2010) 'Loading of the knee joint during activities of daily living measured in vivo in five subjects', *Journal of Biomechanics*, 43(11), pp. 2164–2173. Available at: https://doi.org/10.1016/j.jbiomech.2010.03.046.

Enhancing Accessibility and Sustainability in Smart Cities and Smart Buildings: The Universal Accessibility Suite Initiative

www.euproject-access.eu/en/

@EU project AccesS

@EUprojectAccesS

@EUprojectAccesS

This project has received funding from the European Union's Horizon Europe research and innovation programme under the grant agreement number 101147722. The European Union is not liable for any use that may be made of the information contained in this document, which is merely representing the authors' view.