SAFETY4RAILS

Resilience strategies for multi-modal metro and railway systems

Deliverable 5.2

Lead Author: NCSRD

Contributors: FHG, IC, UREAD, MDM

Dissemination level: Public

Security Assessment Control: Passed

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 883532.

D5.2 RESILIENCE STRATEGIES FOR MULTI-MODAL METRO AND RAILWAY SYSTEMS			
Deliverable number:	5.2		
Version:	1.0		
Delivery date:	04/04/2022		
Deliverable due date:	31/03/2022		
Dissemination level:	PU - Public		
Nature:	Report		
Main author(s)	Christos Kyriakopoulos	NCSRD	*Main author* *Drafting of text, iterative feedbacks, quality assurance, language and consequences assessment.*
Main contributor(s) to main deliverable production			
Internal reviewer(s)	Uli Siebold Atta Badii Stephen Crabbe Antonio De Santiago Laporte	IC UREAD FHG MdM	
External reviewer(s)	Dr. Brigitta Sáfár	Member of External Advisory Board	

Document control			
Version	**Date**	**Author(s)**	**Change(s)**
0.1	21/02/2022	NCSRD	Initial version
0.2	21/03/2022	NCSRD	Write section 1.3 Update section 4.1.3 to reflect latest changes
0.3	23/03/2022	NCSRD	Update section 2.3.2.1 (Ankara 3D model) Minor corrections to MDM and EGO scenarios Removed section 2.2 Added section 2.3 (requirements) Updates after internal review from IC
0.4	28/03/2022	NCSRD	Update chapter 5 Updates after internal review from UREAD
0.5	29/03/2022	NCSRD	Updates after internal review from FHG Move scenarios' and models' details to D5.7
1.0	04/04/2022	NCSRD	Conversion of V0.5 to V1.0 following confirmation of security assessment "passed"

DISCLAIMER AND COPYRIGHT

ABOUT SAFETY4RAILS

SAFETY4RAILS is the acronym for the innovation project: Data-based analysis for SAFETY and security protection FOR detection, prevention, mitigation and response in trans-modal metro and RAILway networkS. Railways and Metros are safe, efficient, reliable and environmentally friendly mass carriers, and they are becoming an even more important means of transportation given the need to address climate change. However, being such critical infrastructures turns metro and railway operators as well as related intermodal transport operators into attractive targets for cyber and/or physical attacks. The SAFETY4RAILS project delivers methods and systems to increase the safety and recovery of track-based inter-city railway and intra-city metro transportation. It addresses both cyber-only attacks (such as impact from WannaCry infections), physical-only attacks (such as the Madrid commuter trains bombing in 2004) and combined cyber-physical attacks, which are important emerging scenarios given increasing IoT infrastructure integration.

SAFETY4RAILS concentrates on rush hour rail transport scenarios where many passengers are using metros and railways to commute to work or attend mass events (e.g. large multi-venue sporting events such as the Olympics). When an incident occurs during heavy usage, metro and railway operators must consider many aspects to ensure passenger safety and security, e.g. carry out a threat analysis, maintain situation awareness, establish crisis communication and response, and they have to ensure that mitigation steps are taken and communicated to travellers and other users. **SAFETY4RAILS will improve the handling of such events through a holistic approach.** It will analyse the cyber-physical resilience of metro and railway systems and deliver mitigation strategies for an efficient response, and, in order to remain secure given everchanging novel emerging risks, it will facilitate continuous adaptation of the SAFETY4RAILS solution; this will be validated by two rail transport operators and the results will support the re-design of the final prototype.

TABLE OF CONTENTS

ABOUT SAFETY4RAILS....3
Executive summary....6
1 Introduction....7
1.1 Overview....7
1.2 Structure of the Deliverable....7
1.3 Task Dependencies....7
2 Role of Simulation in SAFETY4RAILS....8
2.1 Role in Each Resilience Phase....8
2.2 Contribution....8
2.3 Requirements....9
3 iCrowd Simulator....11
3.1 Overview....11
3.2 Architecture....11
3.2.1 Modules....12
3.2.2 Network Communication....14
3.2.3 Embedded Lua Interpreter....15
3.2.4 Visualization....16
3.2.5 Distributed Simulation....16
3.2.5.1 Implementation....16
3.3 Autonomous Agent Simulation....17
3.3.1 Routing via Navigation Mesh....17
3.3.2 Collision Avoidance....18
3.3.3 Behavioural Modelling with Behaviour-Trees....19
3.4 Added Functionality for SAFETY4RAILS....20
3.4.1 Escalators....20
3.4.2 Crowd and object detection....22
3.4.3 Detection Evasion....23
3.4.4 Pressure due to Congestion....24
3.4.5 Native Visualization of Results....24
3.4.5.1 Congestion Levels....25
3.4.5.2 Pressure Levels....26
3.4.5.3 Monitored Areas....27
3.4.6 Integration with DMS....28
3.5 Training....30
4 Integration with Other Tools....31
4.1 BB3D....31
4.1.1 Purpose of Integration....31
4.1.2 Visualization....32

4.1.3 Communication Protocol (API)....32
4.1.3.1 Request Message (iCrowd to BB3D)....32
4.1.3.2 Response Message (BB3D to iCrowd)....33
5 Summary and Conclusion....36
5.1 Future work....36
Bibliography....38
ANNEX I. LIST OF ABBREVIATIONS....40

List of tables

Table 1: iCrowd's contributions per resilience phase....8
Table 2: Development status of iCrowd requirements as per D1.4....10
Table 3: List of abbreviations....40

List of figures

Figure 1: The architecture of the iCrowd simulator....12
Figure 2: A multi-level navigation mesh (blue) on top of a geometry (grey)....17
Figure 3: The PCR of an entity created by other nearby entities....18
Figure 4: The behaviour tree of an agent representing a Border Guard in a Risk-Based Border Crossing Point, with the ability to approach a random agent and stop them....20
Figure 5: Crowd using the correct escalator according to the direction of their movement....21
Figure 6: A guard (blue agent) monitoring an bounded area (marked with green dots, bounded with red ones) with normal agents (yellow)....22
Figure 7: The trajectory (green line) that a malicious actor (red) will follow to evade detection by the guard (blue agent) and the camera (blue box)....23
Figure 8: Example voxel grid visualization from the Proximity Module....25
Figure 9: Heatmap of congestion levels in an underground metro station with a box-outline rendering type. .26
Figure 10: Pressure levels of individual agents in a stampede (yellow means no pressure, red means maximum pressure)....27
Figure 11: Heatmap of detectors' coverage by guards and cameras....28
Figure 12: A GUI window allowing the user to send custom messages to the DMS server....28
Figure 13: The DMS module showing the result of all HTTPS calls made to the DMS server....29
Figure 14: Visualization of BB3D’s results of a bomb explosion....32

Executive summary

This document reports on the results of Task 5.2, which aims to deploy an agent-based simulator to discover possible vulnerabilities of railway and metro stations, estimate parameters useful for risk mitigation strategies, and test the effectiveness of surveillance and security policies.

This deliverable D5.2 is a public report containing the work done on the iCrowd simulator in the context of SAFETY4RAILS. It is targeted to wider audiences, showcasing the capabilities of the iCrowd simulator, along with the SAFETY4RAILS consortium and the European Commission together with evaluators in the context of the SAFETY4RAILS project. The SAFETY4RAILS deliverable D5.7, which is a confidential report, is an extension of this report, containing the actual inputs that were provided by- and outputs that were calculated and provided to the end-users. It targets a smaller audience but its content is complementary to D5.2 and should be considered together with it.

1 Introduction

1.1 Overview

The Description of Action (DoA) describes this deliverable as a “Report on resilience strategies for multi-modal metro and railway systems”.

As a direct result of the work done in the context of Task 5.2, this report presents the use of a real-time agent-based simulator for the detection of possible vulnerabilities of the resilience strategies applied in multi-modal metro and railway systems, as well as the estimation of useful parameters such as the total evacuation time, the average response time, and others.

This report aims to describe the systems, frameworks, and communication devices that were used to implement the above, and present the work that was done to extend them and integrate them in the SAFETY4RAILS project. As the task 5.2 revolved around the use of the iCrowd simulator, the work that is presented in this deliverable focuses on the features that were created or extended to facilitate a set of simulations for the SAFETY4RAILS project.

This deliverable D5.2 is a public report. While it fully describes the systems and logic that were used to evaluate the resilience strategies in question, it does not include actual strategies, raw inputs or outputs, interpreted results, or conclusions in any way or form. The raw inputs, outputs, and results will be part of the deliverable D5.7 which is confidential (only for the members of the Consortium, including the Commission Services). The high-level conclusions drawn from the raw results by the end-users will be available in the deliverables of the respective end-users.

The target audience of this report is the SAFETY4RAILS Consortium and any other external party interested in NCSRD’s iCrowd simulator. The European Commission together with evaluators are also targeted audiences of the deliverable.

1.2 Structure of the Deliverable

This deliverable is structured as follows:

- Section 1 introduces the deliverable, its structure, and its task dependencies.
- Section 2 describes the role and objectives of the simulation platform in the context of SAFETY4RAILS.
- Section 3 presents the iCrowd simulator on which the simulation platform is built, and the extensions that were implemented in the context of this project. This covers the work that has been done in task 5.2.
- Section 4 presents the integration of iCrowd with other tools of the SAFETY4RAILS project.
- Section 5 provides the summary and conclusions of this report, along with potential future work that can be done to further improve the simulation platform.

1.3 Task Dependencies

The deliverable depends on the results of the following tasks:

- Task 1.2: Tool requirements
- Task 1.4: Data collection
- Task 2.3: Overall architecture of the S4RIS platform and its data exchange system
- Task 2.5: Original use-cases
- Task 3.1: Identification and characterization of cyber (-physical) systems and threats in railway environments
- Task 4.5: Cascading effects for interconnected infrastructures, integration with other tools
- Tasks 8.1 and 8.2: Simulation exercises

2 Role of Simulation in SAFETY4RAILS

2.1 Role in Each Resilience Phase

The role of the iCrowd simulator in the context of SAFETY4RAILS is to be used for the detection of possible vulnerabilities in multi-modal railway and metro stations, extract useful metrics for their evaluation, and by doing so provide a risk assessment tool to develop better resilience strategies for the safety and security of the users of such infrastructures.

This is achieved by simulating the impact of cyber-physical threats on metro and railway stations, taking into account the crowd's behaviour, external interconnected infrastructures, simulating realistic information propagation models, and using the infrastructure's current surveillance and security policies, such as evacuation processes, CCTV systems, security personnel positioning, etc.

The iCrowd simulator is used only during the prevention phase of developing and testing resilience strategies, but contributes to more of them by simulating them. A detailed description of iCrowd's functionalities for each of the relevant phases[1] is shown in Table 1. Cross-cutting "mitigation" in Table 1 illustrates the functionality to test different mitigation measures.

PREVENTION	DETECTION	RESPONSE
Predict waiting and service times and detect bottlenecks to determine the infrastructure's ability to handle high congestion levels. Examine the effects of ineffective guards (due to distractions). Verify or disprove the sufficiency of security and/or safety measures in place that prevent accidents or malicious behaviours when guards are unavailable.	Aid in detection capabilities by detecting blind-spots because of guards' movements and/or insufficient number of cameras. Improve positions and movement patterns of guards by simulating a malicious actor accessing a restricted area and determining the average time-to-detect and time-to-intercept.	Simulate an evacuation and examine the crowd's behaviour, positions, and movements in such conditions to determine the best possible movement patterns for security personnel.
MITIGATION		
Simulate an evacuation and assess the performance of resilience strategies applied in such conditions. Measure the efficiency of mitigation strategies that rely on digital assets (e.g. electronic boards).		

Table 1: iCrowd's contributions per resilience phase

2.2 Contribution

As a result of tasks 2.5, 8.1, and 8.2, a set of Use-Cases (UCs) were created in coordination with the consortium, based on which, presently, 2 detailed simulation exercises were defined; the MDM and EGO simulation exercises. Each of these detailed a scenario which would use a subset of the tools participating in the project to demonstrate a set of resilience capabilities.

iCrowd contributes to the development, implementation, and validation of resilience and mitigation strategies by simulating both exercises. The simulation scenarios were based on the top-level actions as they were

[1] In other SAFETY4RAILS documents, resilience is represented by the 5 phases: Identification, protection, detection, response and recovery. In the D8.2 for the planning of the simulation exercise at MdM and EGO the phases identification and protection were simplified into a combined phase "prevention". Considering the fuller 5 phase representation of resilience listed here, iCrowd is most suitable for application during the identification and protection phases.

defined in D8.2, which were followed as accurately as possible. The simulated scenarios regarding input, outputs, programmed flow, 3D models, and contributions, are presented in Deliverable D5.7.

The overall functionality that is provided by iCrowd is the following:

- **Determine infrastructure's ability to handle high congestion levels**: Simulate large crowds entering and exiting the station in normal circumstances and in case of emergency or schedule disruption. Eventually, provide a visualization of the overall movement patterns and congestions levels, all within the graphical user interface of the simulator.
- **Study schedule disruptions' impact on agents' flow rate**: Deactivate the arrivals and departures of trains in a simulation and inspect how the crowd moves around the station during high congestion levels.
- **Detect and reduce crowd bottlenecks**: Simulate large crowds moving in the same direction or towards the same target and visualize the congestion levels and pressure levels using heatmaps, in order to better understand the weak points of the station and better set up the available space.
- **Validate and improve CCTV systems**: Simulate cameras and guards moving around the environment and provide heatmaps and statistics regarding the detection of malicious actors. Simulate malicious actors by implementing real-time dynamic detection evasion and examine the resulting movement trajectories. Provide data to eventually improve the coverage of the available space whilst minimizing the required resources.
- **Validate and improve evacuation processes**: Execute simulations based on real measurements to establish a baseline, and then run more simulations with adjusted parameters to examine alternate evacuation processes.
- **Implement and measure user-defined KPIs**: Provide end-users with realistic measurements for their own KPIs for every strategy they wish to evaluate that may be difficult or even impossible to obtain in real life.
- **Provide a test-bed for the development of security and safety measures and mitigation strategies**: Run time-scaled simulations to visualize the results of the disruption and fallout of an attack, and the application of the studied strategy. Provide an overview of how the scenario unravels in fast-forward, observe the effect of real-time adjustments to the simulation and determine chain reactions. Reduce the cost of developing and testing new safety measures and mitigation strategies by reducing the need for physical trial runs with actual people.

2.3 Requirements

In the following table, the requirements defined in Deliverable D1.4 are listed along with their development status.

Req. ID	Name	Status	Comments
iCrowd_01	Simulate realistic congestion levels	Completed	Available from previous projects
iCrowd_02	Simulate an evacuation because of terrorism or natural disaster	Completed	Available from previous projects
iCrowd_03	Simulate crowd behaviour considering cyber-agents	Implemented but not tested	Scenarios for upcoming simulation exercises should be adapted to include this functionality
iCrowd_04	Detect blind-spots because of guards' movements or insufficient cameras	Completed	See Detection Layer at section 3.4.2
iCrowd_05	Simulate access to a restricted area by cyber-attack or physical attack	Completed	Part of the EGO scenario
iCrowd_06	Guards' distraction simulation	Implemented but not tested	Scenarios for upcoming simulation exercises should be adapted to include this functionality

iCrowd_07	Conformity with overarching and S4RIS platform specific requirements	Completed	iCrowd runs independently from other tools Communication is done through DMS (see section 3.4.6) User-programmable via Lua scripts iCrowd user manual is available

Table 2: Development status of iCrowd requirements as per D1.4

The new functionalities that were developed to cover the above functionalities are described in section 3.4.

3 iCrowd Simulator

3.1 Overview

The iCrowd Simulator [1]–[4] of the National Center for Scientific Research «Demokritos», initially designed and developed in the context of the TASS Project[2], is a general purpose Agent-Based modelling platform aiming to provide an abstract, domain-agnostic simulation framework. It implements a modern, multithreaded, data-oriented simulation engine employing the latest state-of-the-art programming technologies and paradigms.

iCrowd is based on an extensible architecture that separates core services from the individual layers of agent behaviour, offering a concrete simulation kernel designed for high-performance and stability. Its primary goal is to deliver an abstract platform to facilitate the implementation of several agent-based simulation solutions with applicability in many domains of knowledge, including but not limited to crowd behaviour simulation during evacuations and social behaviour simulation and modelling.

Thanks to its high performance parallel execution model, iCrowd is capable of handling very large-scale crowds. It can be utilized in any area such as building interiors and exteriors, stadiums, public places like squares, open-air festivals, or even a small city. It can be deployed on a single system with consumer-grade hardware for simple simulations, or on more advanced clusters to facilitate more complex scenarios in real time. The latter is achieved by distributing parts of the workload (entities, behaviours, physical spaces) across multiple compute nodes.

Some key technical features are the following:

- High performance parallel execution model developed in C++14, with a standards-compliant, cross-platform (Win32, Linux, MacOS) codebase
- Intelligence model using Behaviour-Trees [5]–[8]
- High-precision autonomous movement model based on ClearPath [9]
- Autonomous collision avoidance system based on Velocity Obstacles [10] and social forces [11], [12]
- Extended connectivity support (raw TCP streams, HTTP APIs, Apache Kafka [13] integration)
- Scriptable through an embedded Lua interpreter [14]

To construct the simulation environment, iCrowd uses a 3D model of the infrastructure of interest, given in OBJ [15] file format. There is no need for any annotations or extra information within the OBJ file, as any navigation-related information is automatically calculated. In cases where no specific infrastructure needs to be simulated, for example if only the interaction of the agents with specific assets is of interest, then a simplistic and generic layout can be used instead.

3.2 Architecture

The iCrowd simulator is based on a modular architecture that builds on the Entity-Component design paradigm. This allows the separation of the core services of the simulation engine from the distinct Layers that comprise each agent's profile and behaviour. Thus, the main processing kernel that deals with resource allocation and processing synchronization is separated from the individual behaviour implementations that usually deal with higher level functionalities (steering, pathfinding, intelligence, communications etc.). The simulation engine acts as an orchestrator for the distinct Layers, which may function individually or in cooperation with one another.

Everything that can be regarded as an 'active' element of the simulation is an Entity and has very basic functionality. Complex behaviours are provided by plug-in modules. Even the most fundamental functionalities such as the Physical movement or the Intelligence of an entity are modelled and programmed separately. This is achieved by populating Entities with Components provided by Layers that reside outside of the simulator's core. Each Entity has a set of Components, and each Component is part of a Layer. This is shown in Figure 1.

[2] For more information visit https://cordis.europa.eu/project/id/241905/fr

At every simulation step, each Layer updates itself and all of its Components, independently from other Layers. Synchronization mechanisms are in place to enable Layers or Components to cooperate with each other.

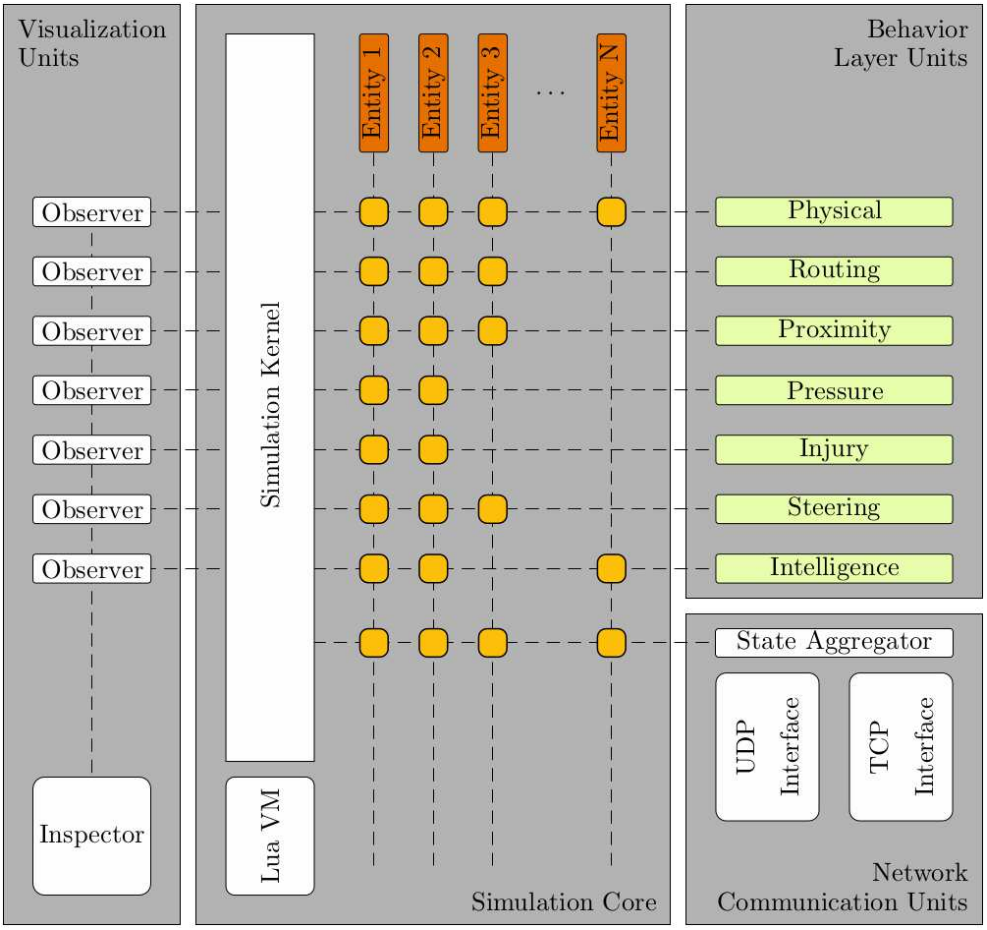

Figure 1: The architecture of the iCrowd simulator

3.2.1 Modules

Entities can have one or more attached Components, provided by the associated Layer. A Layer is a module that implements or retains global functionality or information, while the Components of each Layer implement or retain entity-specific functions or data. The collection of Components for each entity forms the basis of the entity's behaviour and state. Each Layer and its respective Components provide their entities with a different functionality by implementing various functions, keeping any relevant data, and possibly coordinating with each other. Each module can also have a network interface and/or an Inspector Observer. The network interface of a layer is a list of parsers of messages and a list of serializers of events, that can be used to receive or send messages through iCrowd's Network Communication System (see Network Communication). An Inspector Observer is a GUI that is visible to the user in a global menu or a selection window of an entity and is used by the Inspector module (see Visualization). The iCrowd simulator offers the following modules:

- **Physical Layer**: Components of this layer implement kinetic and physical behaviour and are responsible for providing their Entity with its physical characteristics such as mass, dimensions as well as making it behave as a real-world object affected by forces. Entities that represent agents always have a Component of this layer, but entities that represent checkpoints, for example, may not.
- **Routing Layer**: This Layer's components provide routing and path-finding capabilities to their entity as well as bindings for other Components that need to perform such requests. The Layer itself creates and maintains a navigation mesh during initialization and shares it with all its components. The actual path-finding is done on a per-entity basis as the navigation mesh can be locally tweaked by enabling or disabling parts of it on the fly for a specific entity. See Routing via Navigation Mesh for more information.
- **PathBlock Layer**: This Layer's Components keep a navigation filter for their entity, enabling it to block a specific area of the environment for any reason. For example, a passenger in an airport might

be prohibited from moving in a restricted area, so the entities that represent passengers will use the PathBlock Component to block that area. A second example would be that no agent can move in an area because it is on fire. The PathBlock Layer, in this case, will maintain a list of areas that are on fire, and all of its Components will block anything that is on that list. This functionality can be extended by having the Component use this list only when the entity has been notified about the fire.

- **Proximity Layer**: The Components of this Layer gather and maintain proximity-based information about their entities. This type of information includes, for example, a list of other entities which are close-by.
- **Steering Layer**: A Steering Component makes use of its entity's Proximity Component and alters its desired velocity (targeting the entity's final destination) in such a way as to avoid collisions with other, stationary or moving, entities that are nearby. This Layer can also be used to implement social distancing by adjusting its collision avoidance radius and resolution forces. See Collision Avoidance for more information.
- **Injury Layer**: Each Injury Component uses an injury model (currently based on levels of carbon monoxide, soot, and heat) to calculate the physical state (in terms of injuries sustained) of its entity. It can, therefore, alter many of its entity's Physical Component's parameters, such as top speed, navigation abilities, etc.
- **Pressure Layer**: Components of this layer use a pressure model to calculate the total amount of pressure exerted on their entity as well as the pressure transmitted through it to other entities. High levels of pressure can cause injuries to an agent (see Injury Layer).
- **Intelligence Layer**: Components of this Layer run the behaviour tree of their entity. Essentially, Intelligence Components endow the simulator's entities with intelligent behaviour, enabling it to make decisions, set targets, and adapt to their environment. See Behavioural Modelling with Behaviour-Trees for more information.
- **Control Layer**: Components of this Layer provide the user with control over the underlying entities. Such control would be to move them using the mouse or keyboard.
- **Base Communication Module**: This module implements the distributed simulation functionality of the simulator while also enabling external systems to cooperate with iCrowd.
- **Detection Module**: This module implements object detection for entities that represent cameras or guards. It uses an adjustable realistic field of view, and takes into account obstructions from the geometry (such as walls) as well as from the crowd or other entities. This module also records statistics and renders field-of-views, visible walkable areas, and heatmaps.
- **Corridor Module**: This module is used to create entities that agents can walk within, such as stairs and escalators. The module enables an agent to walk faster or slower when walking on an escalator according to its movement direction, while the functionality itself can be enabled or disabled during runtime by the user. The module also collects statistics about such corridors.
- **Checkpoint Module**: This module is used by entities that represent checkpoints, something that can be visited by another entity. It implements a FIFO queue, can be enabled/disabled automatically or manually by the user, and collects statistics about its entities and service history.
- **S4R Module**: This module implements functionality that is specific to the SAFETY4RAILS project, such as the integration with other tools and the visualization of any related information. It also facilitates behaviours that are specific to the SAFETY4RAILS-related scenarios.
- **DMS Module**: This module implements the integration of iCrowd with an Apache Kafka [13] server. It can log in, subscribe to topics, post to topics, and poll for data. It provides a callback mechanism for other modules to receive data, and exposes a simple function for other modules to post data.
- **Inspector Module**: The Inspector module visualizes the simulation by drawing the 3D geometry and entities. It also provides a basic GUI for the user to manipulate the simulation's state, and provides an interface (based on the Inspector/Observer architecture) for other modules to show their own GUI as part of a menu. See Visualization for more information.
- **Visualizer Module**: The visualizer module is only a network interface that provides an API for agent creation, fire creation, environment setup, and agent manipulation during runtime to facilitate the use of an external Visualization Application.

3.2.2 Network Communication

The simulator implements its own Network Communications System, providing bi-directional data communication through TCP or UDP, unicast or multicast. Through this system, iCrowd provides a communication interface, an API, making it possible to integrate and/or interact with other software platforms. Thus, it can be used to provide input to other systems, receive output from other systems (sensors, databases, simulators), or both.

The network communication system implements a simple and flexible JSON protocol. The protocol states that each message starts with a header of constant size which is followed by the actual message. Both the header and the message need to be in a standards-compliant JSON format. The header includes the type and size of the message, informing the iCrowd simulator about the intended recipient (module) of the message. The size of the message is needed because of the way that TCP sockets work. Specifically, we need to know how many bytes of data we want to read from the connection, because there is no other way to know when a message has been fully received. If we attempt to read more bytes than are available, the program will block indefinitely, and if we read less bytes we will receive a malformed incomplete message. The same logic is used in many low-level network communication protocols. Following the header is the message, which can be any JSON-formatted text.

To avoid sending unnecessary data, all JSON objects are not formatted for visibility with spaces and indentations, meaning that there are no spaces or new-line characters between the JSON elements. This is a hard requirement for the header, since it needs to have an exact predetermined size.

A network communication interface is provided by a module of the simulator. This module can also contain a Layer that gets affected by incoming messages or may generate outgoing messages. Upon initialization, a module can be set to function as a server by listening on a TCP port, and can also be set to function as a client by setting a list of hosts and TCP ports it should connect to. Any message that arrives through any of these connections will be parsed by the module, and any event that occurs within the module will be serialized by the network interface and be sent to all of these connections. Ultimately, the network communication interface of a module is described by a list of message parsers and event serializers.

An important communication interface is the one that is provided by the Base Communication module. Its network interface serializes and sends messages for events that include but are not limited to:

- **Pause/Resume**: The simulation has been paused/resumed.
- **Entity created**: An entity has been created. The message includes any information that is necessary for its creation, along with any information that was passed to its components for their initialization, providing support to duplicate the operation in another instance of the simulator.
- **State changed**: A property of an entity has been changed. The message includes the id of the entity, its updated properties, and the timestamp at which the event occurred.
- **Entity deleted**: An entity has been deleted. The message includes the entities id and the timestamp at which the event occurred.
- **Camera view change**: The user has moved the camera. The message includes the new coordinates and view angle of the camera.
- **Time-scale change**: The user has changed the time-scaling of the simulation. The message includes the new time-scaling.
- **Heartbeat**: The time step of the simulation has changed. This is used for synchronization with other systems.

At each time step, each Component of the Layer records the changes that have been done to its entity, serializes them into a JSON packet, and sends it to any connected clients through the network interface. These messages are also retained, so any clients that connect to the simulator after the simulation has progressed will still receive all relevant state updates. The module also sends and receives messages about generic functions like pausing/resuming, adjusting the time scaling, creating/deleting entities, etc. The module's communication is bidirectional, i.e. its outputs can be fed into another instance of itself, achieving a perfect duplication of a simulation's state. See Distributed Simulation for more information.

3.2.3 Embedded Lua Interpreter

As previously stated, iCrowd is domain-independent. Starting the simulator by itself will launch only the core engine of the simulation, without any 3D geometry, layers, or entities. These are all loaded and configured in a script file that iCrowd loads upon initialization. This script is written in the Lua programming language [14], a human-readable language that is designed for scenario scripting and requires little programming experience and knowledge. In this script file the user can load a 3D object to be used as the environment, install and configure any layer they need, install the Inspector if needed, and finally start the simulation. Overall, the functionality of a Lua script includes:

- **Engine configuration**: Settings indicating the simulation's frequency, log level detail, maximum number of entities that can be created per simulation cycle.
- **Modules configuration**: Settings for each module separately. There are global settings that all modules can receive, like tick period, multithreading, etc., and each module can have its own specialized inputs. Every module that provides a network interface can also get settings about its network configuration, namely what port it should listen to as a server, where it should connect as a client, and other specialized settings. The Inspector, being a separate module, is also installed and configured if needed.
- **Operational configuration**: Settings indicating the location of checkpoints, the overall number of agents, when and where the agents should be created, and scheduled events.

Each module exposes its own interface to the Lua scripting mechanism. Within a Lua script, the user can create a module as an object and, after installing it by passing it to a special function, they can keep it for reference to set it up as required and even pass it as a parameter to other modules. For example, the Routing Layer is always kept as a reference and gets passed to other modules that need it. The Intelligence Layer exposes a complete API for each behaviour tree generation, so the user can define their own behaviours directly within the Lua script and attach it to any entity.

The scripting mechanism exposes a schedule function that enables the user to specify an action to happen when the simulation reaches a defined timestamp, and can be used to simulate events, such as a bomb being detonated at a certain point in time, or facilitate a uniform flow of passengers, for example to create an agent every 2 seconds instead of creating them all at once.

Scheduled actions can be recursively rescheduled, so one could create a function that checks for an event and reschedules itself if it has not occurred yet, thus monitor for an event and adapt the scenario accordingly.

The scripting mechanism enables the user to create and set up entities as required, each with their own properties and Components. In the context of SAFETY4RAILS, the user might either be NCSRD or any other partner of the consortium that has been trained by NCSRD as described in section 3.5. A fully functional scenario containing agents, interactive objects, behaviours, and its entire flow can be created from a single Lua script. For the SAFETY4RAILS project, 2 such scripts were created to simulate the full MDM and EGO scenarios as described in Deliverable D5.7.

These scripts included:

- The initialization of the simulation's parameters and modules.
- The creation of static assets, such as stairs, escalators, cameras, and turnstiles.
- The initial and repetitive creation of agents.
- The initial creation of a bomb in a predetermined location.
- A random loitering process for all agents.
- The bomb detonation.
- The random realization of agents in the station that the train is not coming and them exiting the station.
- The blocking of the station's doors and turnstiles.
- The confusion and random movements of agents in the station.
- The release of the station's doors.
- The start of an evacuation.

- The routing of the agents to the safe areas.
- The gathering of statistics.
- The automatic pausing of the simulation when all agents have reached the safe areas.

3.2.4 Visualization

For the simulation inspection and control, the simulator offers an OpenGL-based visualization system with a native GUI based on our Inspector-Observer architecture. The Inspector itself is responsible for window creation and user interactivity, but its main job is the rendering of the 3D environment with all entities that reside in it. The user can use their mouse to move, zoom, and rotate the camera, and can also choose to have the camera automatically follow a certain entity. Using their keyboard or mouse, the user can pause and resume the simulation, change its time-scaling property to fast forward or slow down the simulation, and have a cursor at their disposal which can be attached to any point of the geometry to be referred to by other functions. The Inspector provides a main menu that contains generic settings concerning the visualization of the geometry. It also provides a searchable list of entities that can be used to find and select an entity. Upon selecting an entity, the Inspector provides a selection window that contains all information related to the entity. This includes its name, ID, position, velocity, heading, list of tags, and more. The user also has the option to delete the selected entity with the click of a button.

The Inspector provides any loaded modules/layers to display their options in the main menu and the selection window by implementing and attaching an Observer. Typically, each Layer will have its own Observer which gets plugged in the Inspector upon the Layer's initialization. An Observer is the implementation of at least 2 functions: one that displays a GUI for the main menu and one that displays a GUI for the selection window. There are more functions that an Observer can implement, such as the render function that a Layer can use to render something on the 3D geometry. This is used, for example, by the S4R Layer to render the results of bomb explosions as they are received by the BB3D tool [16] (see BB3D). An Observer can also create and display its own windows if required, as is the case with the Intelligence component which can render the behaviour tree it is using. Usually, these GUIs are only concerned with data from their associated layer or associated entity's component, and can expose adjustable settings or any kind of information that the layer or component has saved to the user. The Inspector provides an easy API to the Observers, allowing them to access information about the selected entity, the cursor position, etc. An Observer can also interact with the Inspector, for example by setting the currently selected entity or the cursor's position.

Overall, the Inspector enables the user to see the simulation and interact with its entities and the underlying components and layers through a simple and easy GUI.

3.2.5 Distributed Simulation

Using iCrowd's Network Communication System, we can connect multiple instances of iCrowd and have them work together [2]. This can facilitate the distribution of the simulation's workload to more than one instance, each one of which can be running on a different machine. The simplest way of work distribution with iCrowd is having one instance run the various modules for a simulation without running the Inspector, and then have another instance connect to the first, fetch the simulation's status updates and visualize it, effectively moving the rendering workload to a different machine. This can facilitate the use of a high performance computer to run the bulk of the computations for the actual simulation and a local consumer-grade system to visualize and inspect the simulation.

This kind of functionality opens the iCrowd simulator to a world of work distribution possibilities, where the workload that is being distributed is not only the visualization of the simulation, but rather entities, layers, or even parts of the 3D space, enabling iCrowd to scale to many systems and potentially simulate hundreds of thousands of agents in real time without the need for a supercomputer.

3.2.5.1 Implementation

To participate in a distributed simulation, an iCrowd instance uses the Base Communication Module which provides messages about entities creation, deletion, state updates, along with more generic events such as pause/resume, time scaling, etc. The information contained in these messages can be used to duplicate an iCrowd instance, and keep all instances synchronized.

The simulation's workload can be distributed statically based on entities. A specialized implementation ensures that only one iCrowd instance will handle the creation, update, manipulation, and deletion of a specific entity, while the rest of them will only retrieve its information to use for collision avoidance, proximity

information, decision making, and visualization. The assignment of entities to instances can be done in various ways, like the role of the entity (human-sensor), the role of the entity as an agent (citizen-civil protection), or randomly according to the time of creation, to name a few. This will distribute the simulation's workload based on its entities and will usually achieve the best load balancing. Alternatively, we can assign entities to iCrowd instances according to the entity's position within the 3D geometry. This would be useful in a case where each instance visualizes only one part of the geometry and focuses on that, while also rendering the rest of the crowd and interacting with it.

Any number of simulation instances that work together need to be synchronized during the entire scenario unfolding. It is expected that no two will run naturally at the exact same simulation time for various reasons (different machines, loads, specs, to name a few). As a result, to achieve a realistic simulation, we need to ensure that all instances are always synchronized, i.e. running at the same timestamp. The implementation is quite simple as the protocol supports bidirectional messages, and consists of a set of primitive rules: (a) Send a heartbeat message periodically; the smallest the interval, the better the resolution; (b) receive heartbeat messages from other instances (c) If a received heartbeat carries a timestamp smaller than the host's current timestamp, pause host for a time period equal to the difference of received timestamp and current timestamp. Implementing the algorithm, all simulators will always operate in sync throughout the entire execution.

3.3 Autonomous Agent Simulation

Every agent within an iCrowd simulation is completely autonomous. It can move, navigate, avoid obstacles, set targets, and interact with its environment or other agents, without any input from the user. This concept is implemented by the functionalities described in the following sections.

3.3.1 Routing via Navigation Mesh

The Routing Layer provides entities with their own autonomous navigation system. The whole process is based on building a navigation mesh. A navigation mesh is a set of interconnected polygons spanning the entire walkable area of a given 3D mesh. The generation of the navigation mesh is done automatically based on the geometry that has been given as input to the simulator, taking into account agents' properties such as their height and radius along with other settings such as the maximum allowed slope and climb. Once the navigation mesh has been built, a graph is generated with a node for each polygon and edges from that node to all other nodes representing polygons adjacent to the initial polygon. One can then route from any point

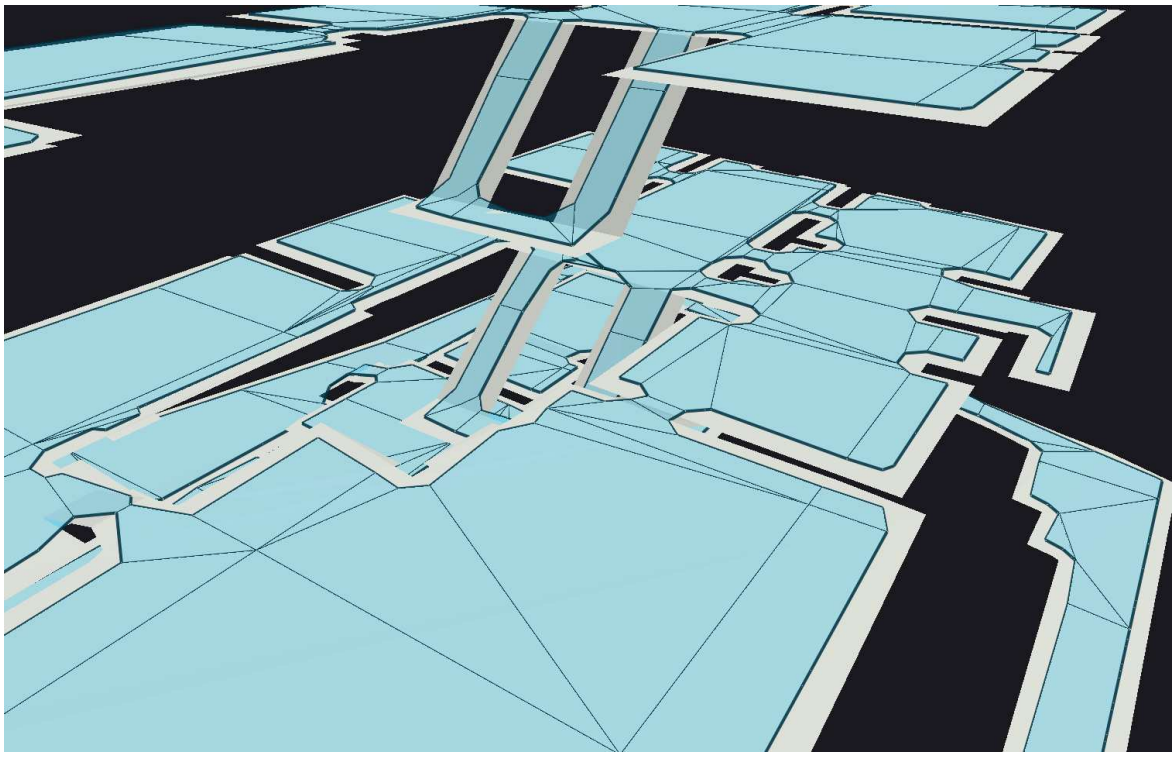

Figure 2: A multi-level navigation mesh (blue) on top of a geometry (grey)

inside a polygon to any other point inside a polygon by running the A*-algorithm [17] on the graph and producing a sequence of waypoints to be followed. In Figure 2 we can see a sample of a navigation mesh as it is constructed by the iCrowd simulator.

Each agent generates a sequence of waypoints to be followed when a target is set and then, at each time step of the simulation, sets its desired velocity directly towards its next waypoint. This velocity will probably be adjusted by the Steering Layer to avoid local collisions but the agent will always try to reach that waypoint. Every agent periodically refreshes their route to account for new information.

3.3.2 Collision Avoidance

iCrowd provides a realistic simulation of humans' physical movement in an area with obstacles and low/high crowd congestion, by implementing a collision avoidance mechanism and taking into consideration social forces that naturally apply when someone is moving among other people. This is achieved by a model of movement for each agent, which is implemented by applying standard Newtonian Forces and interactions between agents, consisting of the Social Forces model [11], [12]. Initially we used the Social Forces model which roughly applied Newtonian forces on each agent, calculated the acceleration from their sum and integrated to get the agent's new position. The additional feature of our model is that apart from applying forces on agents with respect to other agents' positions, we linearly interpolate – under specific conditions – with a force originating from the general movement of agents nearby.

Our current movement model is based on the ClearPath model [9]. The model in question is based on the **Velocity Obstacle (VO)** [10], [18] of one agent onto another. The Velocity Obstacle $VO_A(B)$ of agent A because of agent B is the set of all velocities which will result in a collision with B at some time in the future. A VO depends on both agents' positions and velocities. An agent can calculate one VO for each agent within a certain distance of them. The union of all VOs of an agent is its **Potentially Colliding Region (PCR)** [9]. Simply put, the PCR is the set of points that, when the entity approaches will cause a collision with a nearby entity in the near future. It can be shown that if an agent's desired velocity rests outside its PCR then no collision will occur for that agent with its nearby agents in the near future. If, however, its desired velocity resides inside a PCR, a collision is imminent and the agent must alter its velocity. At this point, the agent will adjust its velocity to the closest point that resides on the border of its PCR.

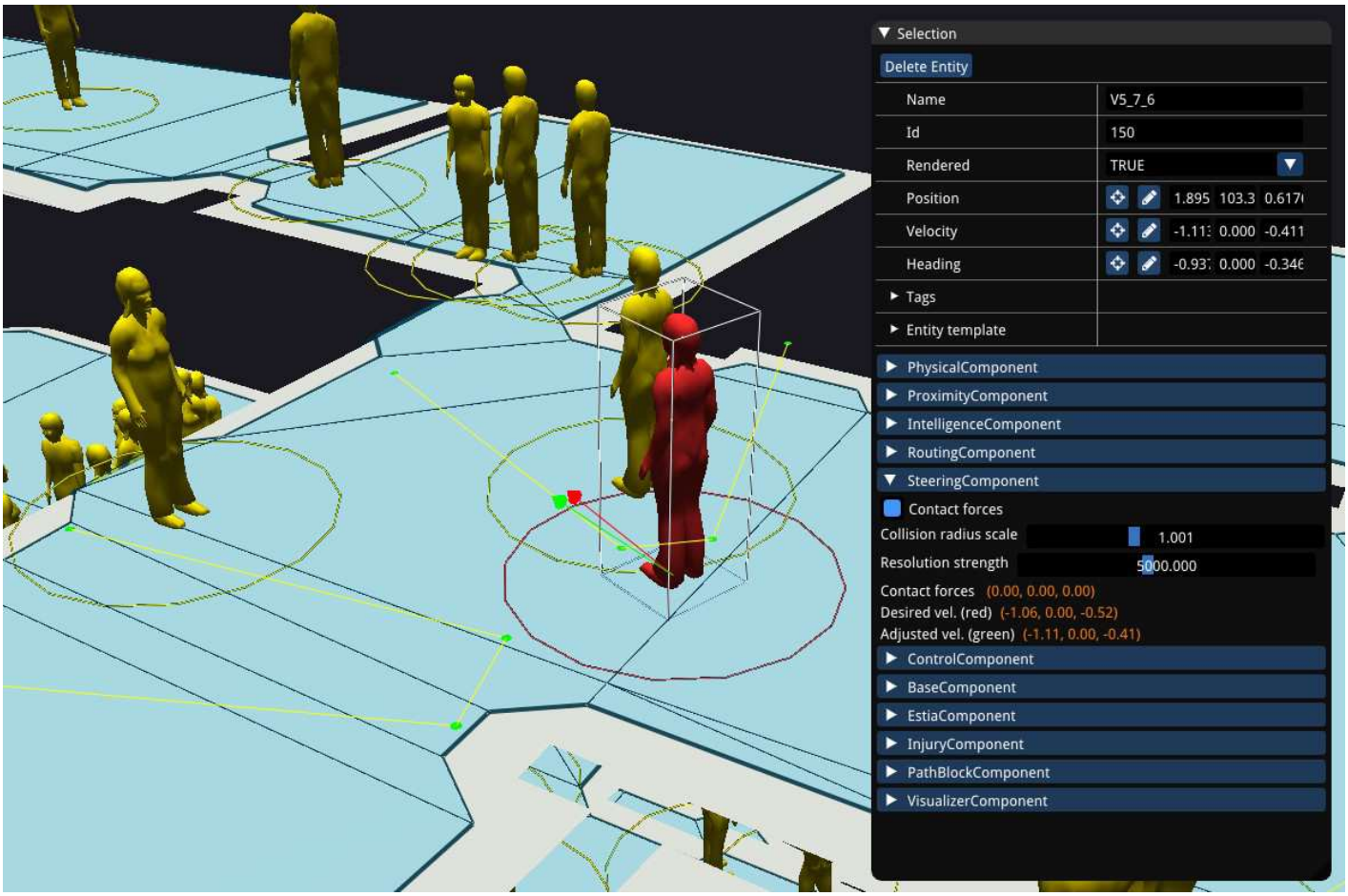

Figure 3: The PCR of an entity created by other nearby entities

The above functionality is provided by the Steering Layer. Each agent considers the current position and velocity of nearby entities to proactively adjust its course to avoid a collision, providing the agents with the ability to walk around the simulated area autonomously, avoiding each other and any static obstacles. The

Steering Layer also allows the user to adjust the collision radius and collision resolution forces. This can be used to tune the Layer itself, but can also be utilized to implement **social distancing**, because with a higher collision radius the agents will be forced to stay further away from each other. This can be useful for use-cases that involve the transmission of a contagious disease and can facilitate such research questions.

In Figure 3, we can see, drawn by the yellow cones, the Potentially Colliding Region (PCR) that has been calculated for the selected (red) entity by its Steering Component. Based on the positions and velocities of the 2 nearby agents (yellow), the 2 VOs that are shown as separate cones have been calculated for the selected agent. The cone on the right is related to the yellow agent inside it and the way it was calculated is obvious. The cone on the left, however, appears to be detached from the yellow entity that caused it. This is because, for the calculation of the VO, we consider not only the position, but the velocity of the agent as well, so since the agent is moving towards the left of the red agent, its VO has been adjusted accordingly.

In the selection menu on the right of Figure 3 we can see the desired velocity that has been calculated by the Routing Component, and the adjusted velocity which has been calculated by the Steering Layer. The two velocities are also rendered as arrows originating from the position of the entity, with red being the desired that will eventually cause a collision, and green being the adjusted one that will be used. We can see that the desired velocity is within the PCR, and the adjusted velocity is exactly on the PCR's border.

3.3.3 Behavioural Modelling with Behaviour-Trees

The iCrowd simulator is equipped with the Intelligence Layer, which uses behaviour trees [5]–[8] to accurately describe the behaviour of an entity, giving it a notion of intelligence in the form of an interdependent cycle of goal setting, succeeding or failing. These were first developed for games and were later formalized using standard robot control theory tools. In general, a behaviour tree is a directed tree with one root node. Each node can be of two types: decorator (or internal) and leaf node. Decorators specify the order and type of execution of their children and leaf nodes specify actions. For our model, we followed [8] in their formal definition of behaviour trees.

The execution of a behaviour tree starts from the root which sends ticks with a certain frequency to its child. A tick is an enabling signal that causes the execution of a child and within the simulation environment follows the simulation frequency. When the execution of a node in the behaviour tree is triggered, it returns to the parent a "running" status if its execution has not finished yet, "success" if it has achieved its goal, or "failure" otherwise.

Behaviour trees can be combined, and there are special types of trees (nodes) that can combine other trees to express more complex meanings. Those combinators are the following:

- **Sequence**: Ticks all of its children sequentially until one of them fails. When one fails, the Sequence block fails. If they all succeed, the Sequence block succeeds.
- **Select**: Ticks all of its children sequentially until one of them succeeds. When one succeeds, the Select block succeeds. If they all fail, the Select block fails.
- **Repeat Until Fail**: Ticks its only child until it fails. The Repeat Until Fail block always succeeds.
- **Repeat Until Success**: Ticks its one child until it succeeds. The Repeat Until Success always succeeds.
- **Parallel Sequence**: Ticks all of its children in parallel, until one of them fails. When one fails, the Parallel Sequence block fails. If they all succeed, the Parallel Sequence block succeeds.
- **Parallel Select**: Ticks all of its children in parallel until one of them succeeds. When one succeeds, the Parallel Select block succeeds. If they all fail, the Parallel Select block fails.
- **Invert**: Ticks its only child until it finishes and returns the opposite result.
- **Succeed**: Ticks its only child until it finishes and then succeeds.
- **Fail**: Ticks its only child until it finishes and then fails.

The combination of any number of distinctively defined behaviours in a well-defined yet flexible sequence results in behaviour models that are reusable, easily extendable and modular. This hierarchical composition of behaviours also allows for a user-friendly and intuitive representation; the behaviour-tree diagram of a high-level behaviour is composed of a behaviour set of lower complexity which is abstracted at the high level diagram.

Essentially, Intelligence Components endow the simulator's Entities with intelligent behaviour, enabling them to make decisions, set targets, and adapt to their environment. Each entity has its own behaviour tree and thus can have completely different behaviour from the rest of the entities. Additionally, a single behaviour tree may be traversed in many ways, depending on its structure and the simulation's state at any time. Agent might, for example, decide to walk to a point of interest and wait there, or find the closest safe area and flee there. This will be decided at the moment that it is needed, depending on whether the agent knows about an evacuation or not. Using parallel decorators, a process such as waiting can be interrupted by events, so an agent standing in a point of interest may abandon it early because they were just notified about an evacuation.

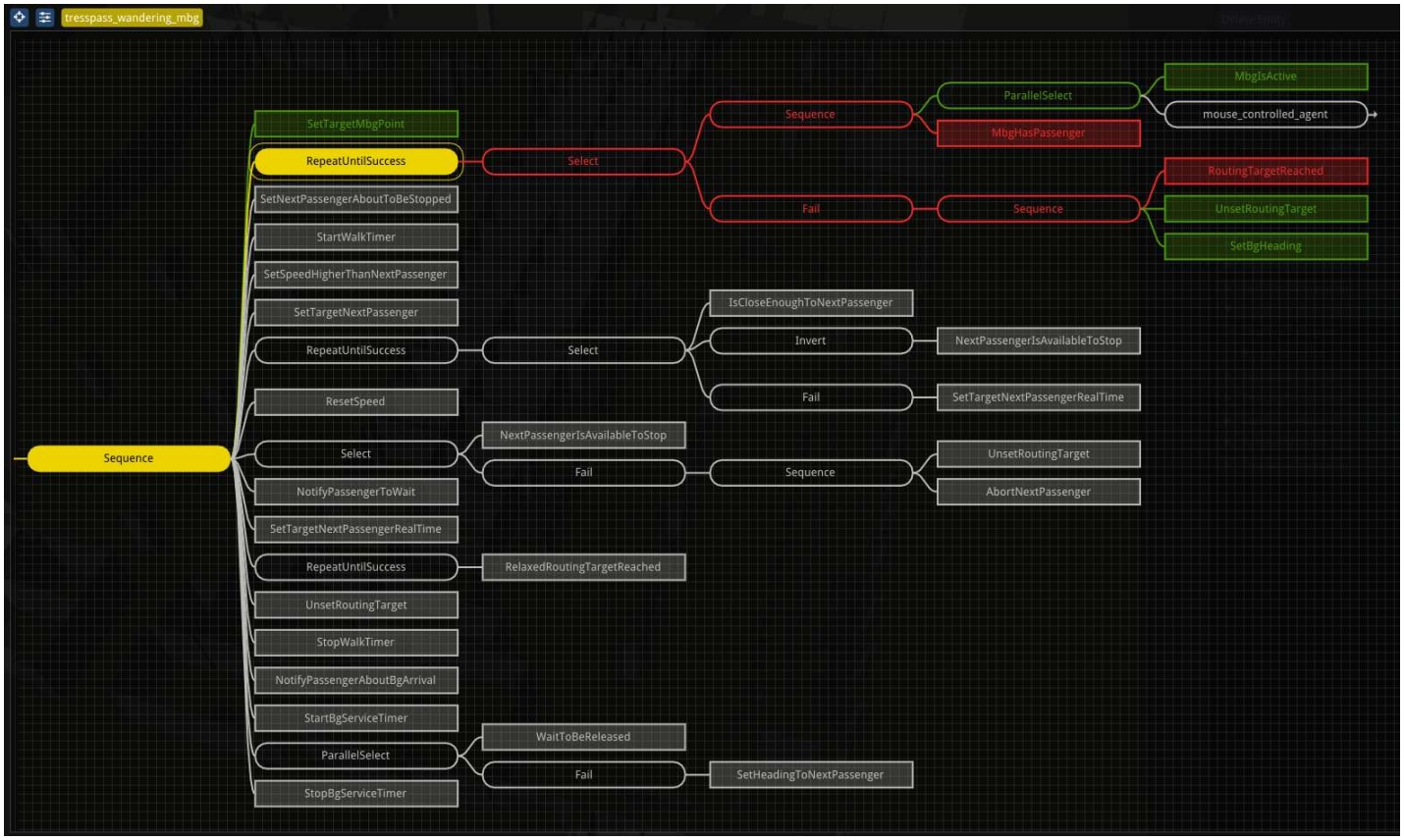

Figure 4: The behaviour tree of an agent representing a Border Guard in a Risk-Based Border Crossing Point, with the ability to approach a random agent and stop them.

An entity utilizing the Intelligence Layer, being either an agent or a stationary object, can perform actions, programmed either by the developers of the platform or by the end user. Using behaviour trees, we can "program" the agents to make decisions based on their interaction with other entities and then move accordingly. Each entity in the simulation can have a different behaviour, which enables us to program different types of people (such as general crowd, officers, guards) or different types of objects (such as sensors, doors, etc.). In Figure 4, a behaviour tree is shown that is used by a Border Guard in a security-sensitive environment, taken from the TRESSPASS[3] project.

3.4 Added Functionality for SAFETY4RAILS

To implement the scenarios required for the SAFETY4RAILS project, iCrowd needed to be extended to cover new functionalities as they were either defined in the tool requirements of D1.4 or needed by the scenarios as defined in D8.2. The new functionalities are described in the following sections.

3.4.1 Escalators

The 3D models used for the SAFETY4RAILS simulations feature directional automatic escalators with a staircase between them. Agents moving between floors should use the escalators and stairs as a real person would. This means that the escalators should be favoured when the congestion levels allow it, and the agent

[3] For more information, please visit https://www.tresspass.eu/The-project

should always use the escalator that is moving in the desired direction, while also adjusting its speed accordingly.

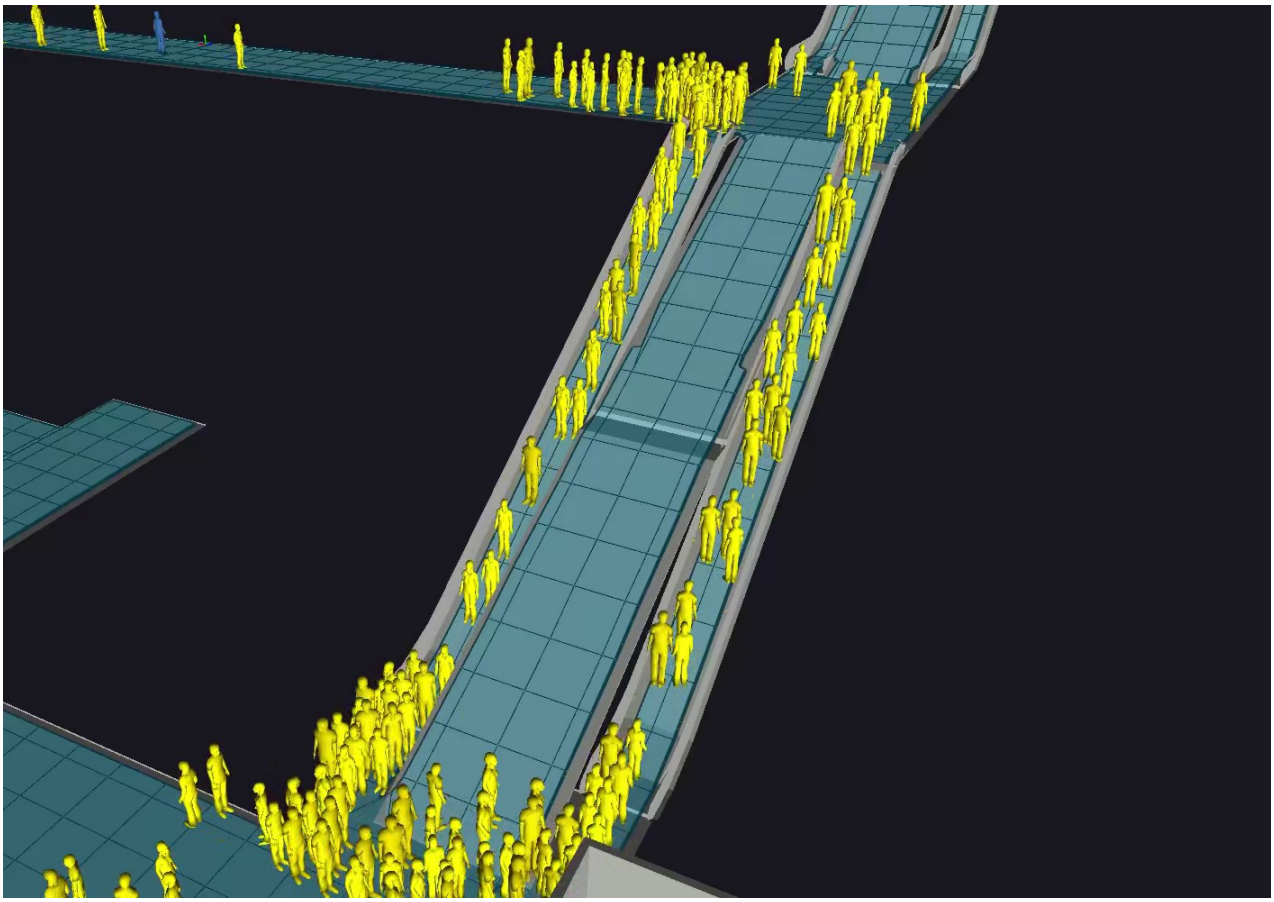

Figure 5: Crowd using the correct escalator according to the direction of their movement

To enable this functionality, the Corridor Layer was developed. Components of the Corridor Layer are attached to entities to make them behave like escalators. The Corridor Component of an entity is associated with a set of navigation mesh polygons, specifically the ones that are exactly below it. When the escalator as an entity is enabled, its Corridor Component coordinates with the Routing Layer to set the associated navigation mesh polygons to have the correct desired direction and a preset penalty for traversing them in the wrong direction. This enables agents to assume that traversing those polygons in reverse would have a great cost, in contrast to traversing them correctly which would have a lower cost than normal. By adjusting the navigation mesh, we are effectively adjusting the graph on which the Routing Layer runs the A* algorithm (see Routing via Navigation Mesh), and therefore taking advantage of the already built in functionality.

An escalator entity also sets a reverse direction speed factor on its navigation polygons. This causes agents that are on an escalator to adjust their speed according to its direction. If they are following the operational direction of the escalator, then the agent's speed is increased, thus simulating a person moving faster because they are on an automatic escalator. If the agent is moving in the reverse direction, then its speed is significantly decreased, to simulate a person trying to counter the reverse movement of the automatic escalator.

Finally, since a corridor can represent any walkable entity, it is also used to implement blocked turnstiles. Even when the turnstile is an active entity represented by a Checkpoint Component, it also needs to be blocked to simulate, for example, a physical or cyber-attack. In this case, no direction is set to the appropriate polygons, but only a constant speed factor. When the turnstile is set to enforce its speed factor, agents going through it are forced to walk slower, to simulate them trying to go over it.

All corridors can be deactivated during runtime by the user or as a result of an event. At this point, the area covered by the corridor can be used normally by the agents. This means that a deactivated escalator can still be used like normal stairs, by resetting the traversal cost of its navigation polygons and providing no speed adjustment. A deactivated pair of escalators will also not be chosen according to their operational movement direction, since it makes no difference if it is stopped.

The escalators' functionality was developed in order to provide realistic simulations in the models that were used for the SAFETY4RAILS simulations. Although not mentioned in this detail in any of D1.4's requirements, this functionality was necessary to ensure the proper utilization of all of our models' assets.

3.4.2 Crowd and object detection

An important functionality that was added to the iCrowd simulator is the crowd and object detection to cover the tool requirement iCrowd_04 (blind spots) from D1.4. Entities that represent static cameras or even moving agents can now access a list of entities that are visible by them based on a field of view, maximum distance, and obstructions. This functionality is implemented in the Detection module, so an entity that can detect visible entities is called a "detector".

A detector continuously watches for other entities (agents or objects) and reports them to the simulation core via a callback mechanism. Each detector has its own position, heading, field of view, and maximum detection distance. These can be set during initialization and can be adjusted in real time, either automatically, based on an event, or manually by the user. Walls and other entities are considered obstacles and may affect the field of view of the detector. In congested environments or complicated geometries with multiple walls, the detector's performance can be realistically reduced.

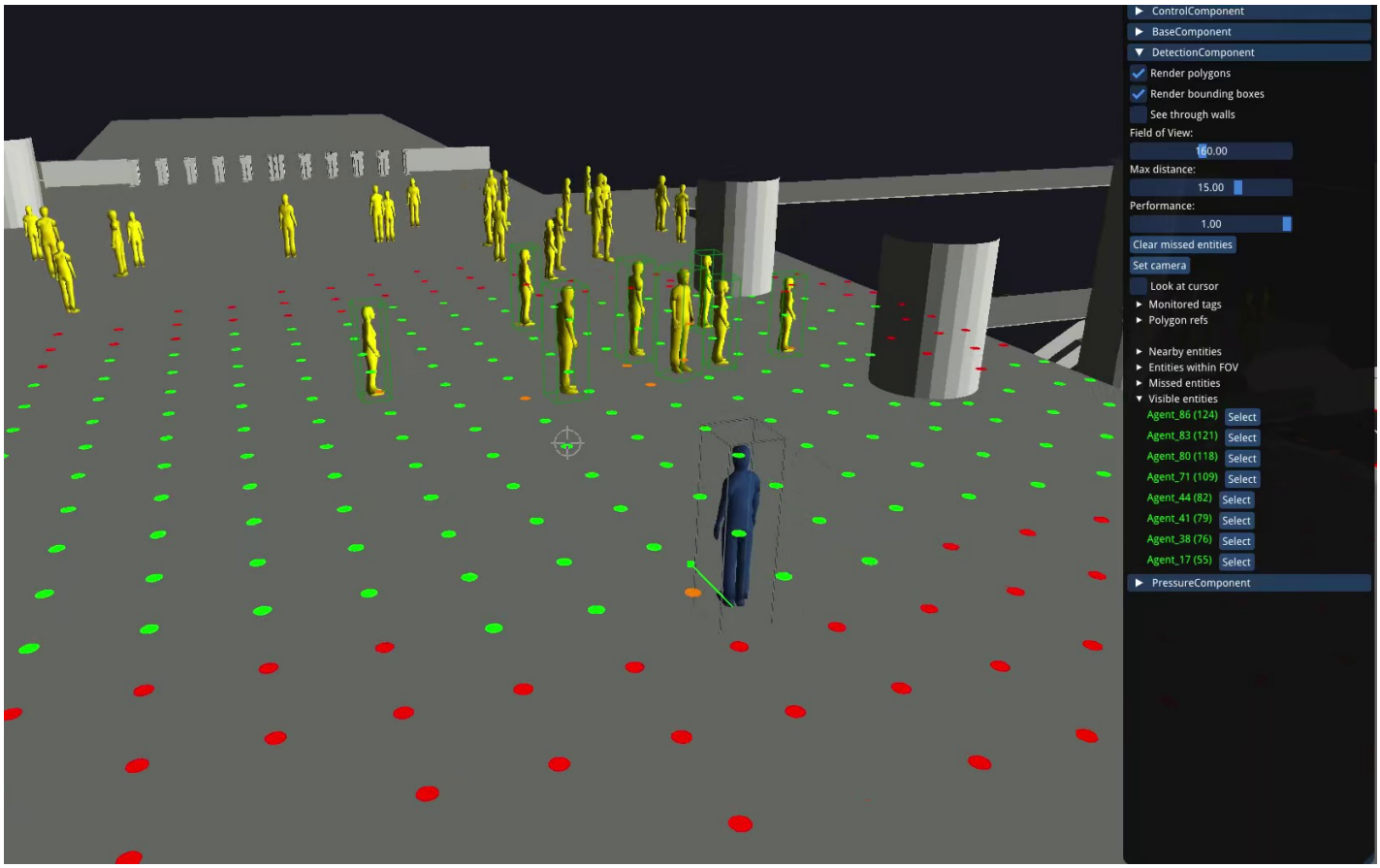

Figure 6: A guard (blue agent) monitoring an bounded area (marked with green dots, bounded with red ones) with normal agents (yellow)

The detection process for each agent is done in steps:

1. The set of entities within a radius equal to its maximum detection distance is retrieved using the Proximity Layer.
2. The entities are filtered based on their tags, allowing a detector to be set to detect only a certain kind of entity such as agents, or malicious actors. This can reduce the computational overhead of the simulation.
3. The rest of the entities are filtered based on the field of view of the detector. This is modelled using the angle between the heading vector of the detector, and the vector that points to the entity starting from the detector's point of view.
4. The remaining entities are filtered for obstructions.

- Other nearby entities are first checked to see if they obstruct the detector's view to a specific entity.
- If not, the geometry is checked.

5. All registered callbacks are called about entities whose visibility status changed (visible -> not visible, and vice versa), and the detector's list of visible entities is updated.

During and after the simulation, iCrowd provides a visualization of the coverage over time or overall, in order to improve the positioning of CCTV cameras or the movement patterns of guards. This is described in detail in section Monitored Areas.

The development of the Detection module aimed to satisfy the "iCrowd_04" requirement as per D1.4.

3.4.3 Detection Evasion

So far, iCrowd has been simulating malicious actors by creating specialized behaviours to perform certain actions. In the real world, a malicious actor might choose to avoid areas that are being monitored by cameras or guards in order to avoid being detected. In the context of SAFETY4RAILS, the EGO simulation exercise (see Deliverable D5.7) features malicious actors who are attempting to breach a restricted zone. In that exercise, iCrowd is employed to determine a CCTV configuration which would adequately detect a malicious actor attempting such a breach. To implement this, we needed the feature described in the previous section, but we also needed to program the malicious actors to attempt to evade detection.

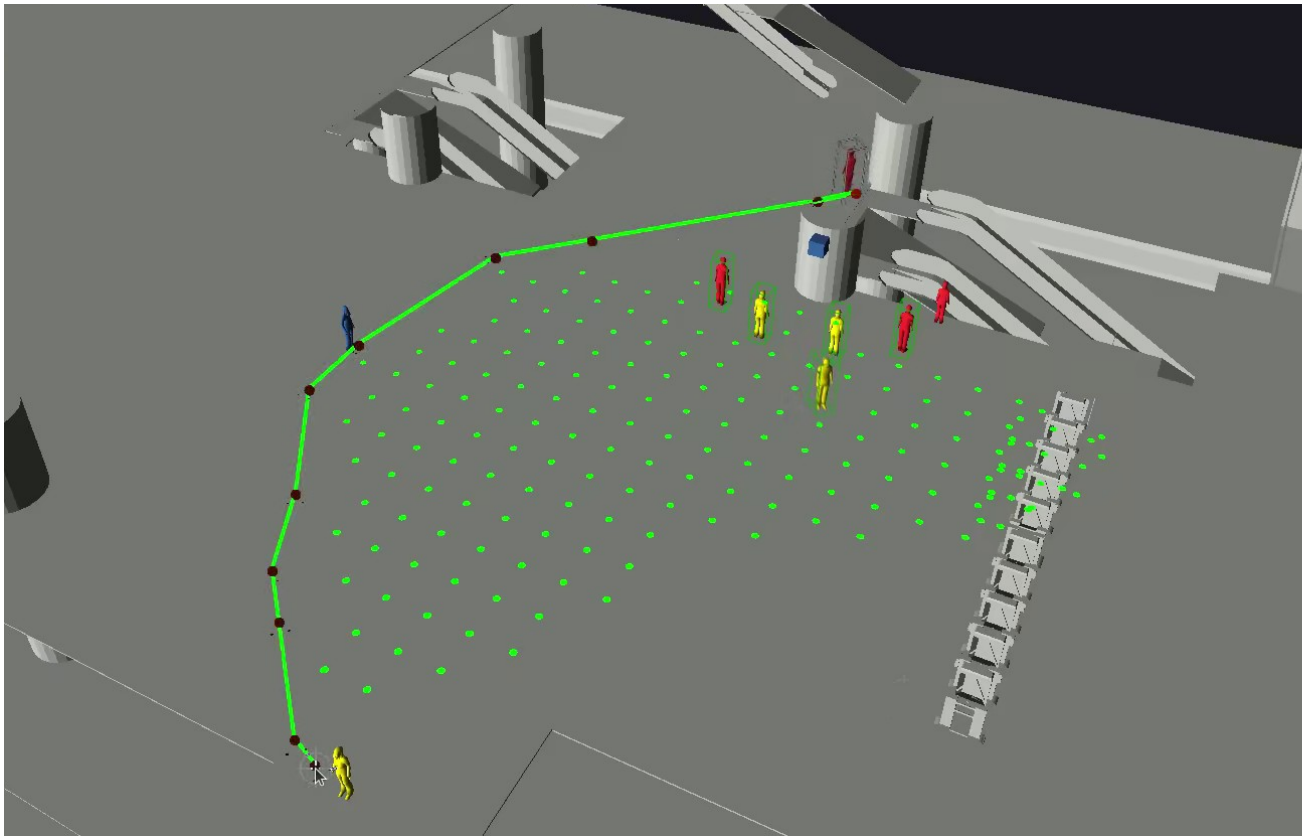

Figure 7: The trajectory (green line) that a malicious actor (red) will follow to evade detection by the guard (blue agent) and the camera (blue box)

To facilitate the functionality of detection evasion, the Detection module was extended. The Detection Component of each detector (be it a camera or a guard) continuously marks all navigation polygons that are visible to it. At each time step, a detector traverses the navigation mesh starting from the polygon that is closest to it, and marks polygons that are within its field of view and are not obstructed by the geometry or other entities. According to the polygon's congestion level (the number of entities currently in it divided by its area), a polygon might be marked as "partly visible", thus making it more likely to be used by an agent who is trying to evade detection and has no better choices. Navigation polygons are easily marked by assigning flags to them, so the Detection module marks visible polygons by assigning the "monitored" flag to them.

Using the already developed PathBlock module, an agent can block the “monitored” flag in order to avoid being routed through areas that are visible by at least one detector. Since this process is based on routing, and agents are regularly rerouted to account for changes in the environment, the process of evasion is continuous; it repeatedly adapts the path of the malicious actor to avoid not only static cameras, but also moving detectors such as guards. If there is no path that avoids detection, the malicious actor will take the path that remains in a monitored area for the least distance.

By setting an agent to avoid the marked polygons, iCrowd simulates a malicious actor who is trying to avoid being detected by cameras or guards, by continuously adjusting its path, even taking into account the congestion levels of an area. Finally, the user of the simulation can inspect the movement trajectory of a malicious actor and, in combination with the coverage heatmaps provided by the Detection Layer, can improve the configuration of the CCTV system or the movement patterns of guards.

The extension of the Detection module was done to enable more realistic behaviours for malicious actors and to provide more outputs for the functionality defined in the “iCrowd_04” requirement as per D1.4. An example of this functionality is shown in Figure 7, where the selected red agent at the top, representing a malicious actor, wants to walk to the yellow agent at the bottom. The blue agent on the left and the blue box on the column next to the malicious actor represent a guard and a camera, respectively. The green dots on the ground show the union of the visible areas of the guard and the camera. The green line shows the path that the malicious agent will follow, in order to avoid walking through the visible areas. If the guard starts moving, then the malicious agent will adapt its path using the updated visible areas.

3.4.4 Pressure due to Congestion

The iCrowd simulator uses its previously developed Pressure Module to calculate a pressure level for each agent in the simulation. This pressure level is calculated based on its proximity with nearby agents and their velocities. Pressure forces are generated by every agent with a non-zero velocity, and are propagated through them when an agent cannot act on a received pressure (move away from it).

The Pressure Layer can be used to calculate the forces applied to agents in stampede situations, where a crowd pushes against a bottleneck of the geometry or a closed door, which causes the agents in the front to experience high levels of pressure. This is visualized in real-time within iCrowd (see Visualization) as changes in the agents’ colours, ranging from yellow (default colour – no pressure) to red (highest pressure), as shown in Figure 10.

In order to cover the requirements “iCrowd_01” and “iCrowd_02” as per D1.4, the Pressure module was extended in order to provide the user with an overview of pressure levels, both during and after the simulation, and to be able to save this information for post processing. At each time step of the simulation, the pressure of each agent is now recorded and can be viewed by the user in iCrowd’s GUI. The maximum and average pressure levels of agents in each area of the environment are also measured and recorded, and can be visualized as heatmaps, as described in Pressure Levels.

3.4.5 Native Visualization of Results

All information that is calculated and recorded by iCrowd can be inspected within its own GUI, as described in Visualization. Statistics that are singular values can be displayed as simple numbers in the respective menus, while others that might be time-series that are sampled throughout the simulation can be rendered in graph format within the native GUI. iCrowd can also record trajectories that can be displayed on top of the geometry. The runtime states of an agent can be visualized as changes in its colour (e.g. pressure level), as a dynamically coloured and sized disk below it (e.g. numerical value retrieved from external software), and as a bounding box around it (e.g. to indicate that an agent is currently visible by a camera).

In the context of SAFETY4RAILS a mechanism for recording and visualizing heatmaps was developed, which can be used to inspect various information within the simulation. This mechanism is based on **3D voxel grids**. A voxel grid covers a subset of the 3D model of the simulation which it divides uniformly across each axis to form individual sub-areas called voxels. Each voxel is a rectangular box with a predefined size, which contains information about the area it covers, such as a temperature, congestion level, pressure level, etc. iCrowd implements 2 types of voxel grids: A vector voxel grid which creates all voxels at initialization and offers quick and easy access to them, and a sparse voxel grid which creates voxels on demand to support larger grids that may cover an entire 3D model. Each module of the simulator can create as many grids as necessary to record and visualize any required information.

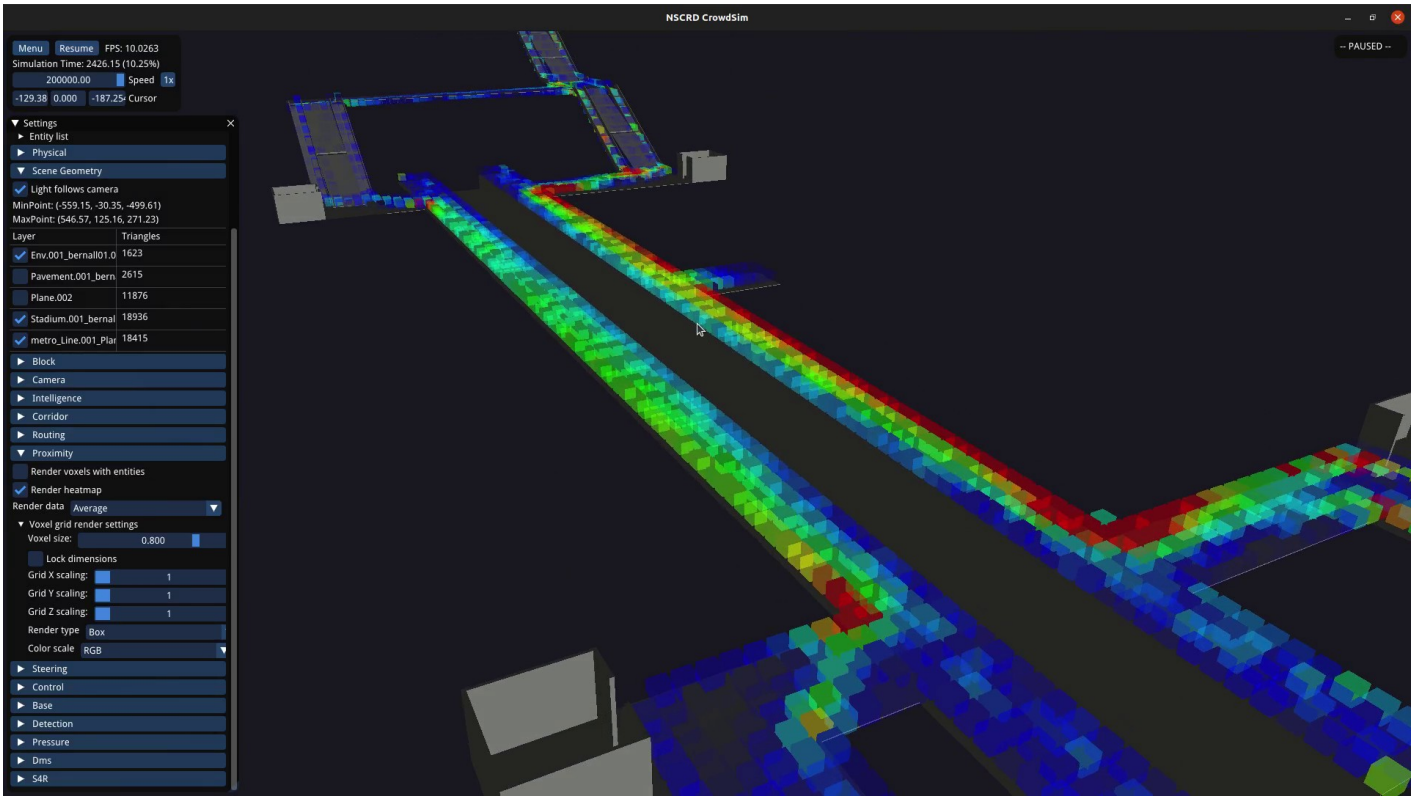

Figure 8: Example voxel grid visualization from the Proximity Module

iCrowd also offers a unified way of visualizing voxel grids, since they all provide a common interface. For each voxel grid that is offered by the simulator's modules, the user can adjust its visualization using a variety of colour scales, scaling options, rendering shapes such as box, box outline, top view, etc. Each module defines its own voxel rendering function which, for each voxel, returns a value in the range [0, 1] that represents its value. This value is then used to calculate the colour of the voxel based on a colour scale (RGB, white-black, etc.), where in an RGB scale, for example, a value of 0 is represented by the blue colour, a value of 1 with red, and all intermediate values smoothly go through green, yellow, and orange. The user can also choose to aggregate a voxel grid during visualization, to view larger voxels with summarized information.

This is linked to requirements "iCrowd_01", "iCrowd_02", and "iCrowd_04", in the sense that, while the requirements were already met, we needed a better and unified way of visualizing their results.

3.4.5.1 Congestion Levels

One of the most used features of the iCrowd simulator is the calculation of realistic congestion levels that can aid in the detection of bottlenecks and "hot" areas, which can have different meanings according to the application. The congestion level of an area, measured as the "number of agents divided by the size of the area" is something that can be immediately apparent to the user through the visualization of the agents themselves, albeit without an absolute numerical value. However, the overall congestion levels of each area are not directly apparent or deducible, and none of this information has any way of being saved and inspected after the simulation.

To facilitate the above needs, the **Proximity Module** of the simulation has been extended to maintain a sparse voxel grid covering only the walkable surfaces of the environment, where each voxel contains the history of congestion levels for the area it covers. For performance purposes, each voxel also keeps the maximum congestion level and the sum of all samples that can be used to calculate their average value since the number of samples is known. The module also keeps a history of the maximum congestion level over the entire model, to enable the proper scaling of congestion levels at all previous timestamps.

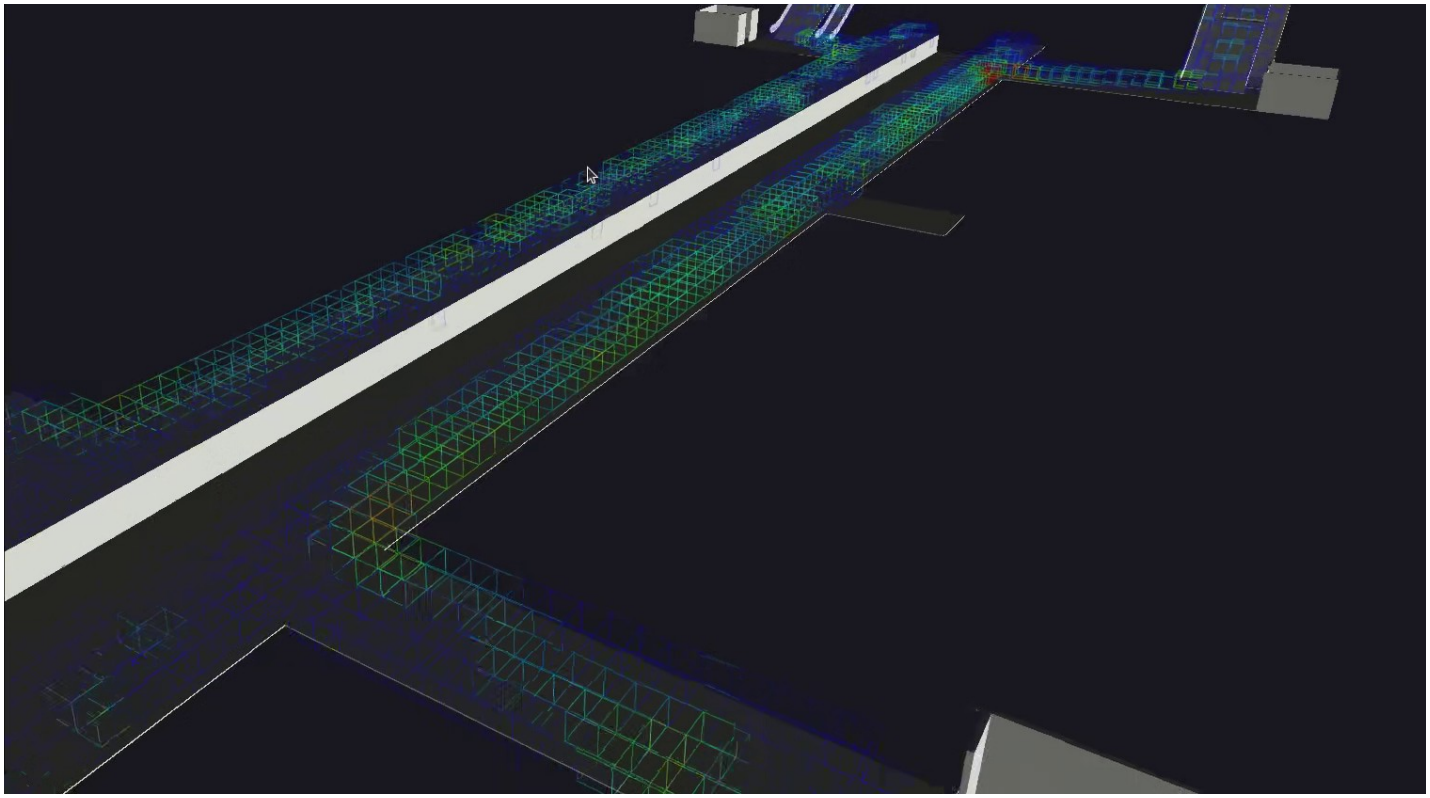

Figure 9: Heatmap of congestion levels in an underground metro station with a box-outline rendering type

The voxel grid that is maintained by the Proximity Module enables a colour-scaled heatmap-type visualization of congestion levels covering the entire 3D model. The user can render the following information provided by the proximity voxel grid:

- **Maximum congestion level**: The value of each voxel is the overall maximum congestion of that voxel divided by the maximum respective value of all voxels. As such, the voxel that has had the maximum overall congestion level across the entire simulation has a value of 1, and all others have a lower value. The values are normalized by the overall maximum congestion value during the simulation.
- **Average congestion level**: The value of each voxel is the overall average congestion of that voxel divided by the maximum respective value of all voxels. As such, the voxel that has had the maximum average congestion level across the entire simulation has a value of 1, and all others have a lower value. The values are normalized by the overall maximum average congestion value during the simulation.
- **Congestion level at a specific time step**: Here, the user selects an earlier time step of the simulation and sees a heatmap of the congestion levels at that time step. The value of each voxel is its congestion level at that timestamp is normalized by the maximum congestion level of all voxels at the same time step. A time scaled playback function is also available showing how the crowd moved throughout the environment.

Other rendering options such as the colour scale, voxel scaling, shape, and aggregation level across each axis are also available to the user, as part of iCrowd's unified voxel grid rendering mechanism.

The extension of the Proximity Module was done in the context of the "iCrowd_01" and "iCrowd_02" requirements as per D1.4.

3.4.5.2 Pressure Levels

Similarly to the Proximity Module, the **Pressure Module** has also been extended to keep a sparse voxel grid covering only the walkable surfaces of the environment, where each voxel contains the history of pressure levels for the area it covers. The pressure voxel grid can provide a colour-scaled heatmap-type visualization of pressure levels covering the entire 3D model. The user can render the following information:

- **Maximum pressure**: The value of each voxel is the overall maximum pressure level of that voxel divided by the maximum respective value of all voxels. The value is normalized by the overall maximum pressure during the simulation.

- **Overall Average pressure**: The value of each voxel is the overall average pressure level of that voxel divided by the maximum average pressure level. The value is normalized by the overall maximum average value during the simulation.
- **Current Average pressure**: The value of each voxel is the overall average pressure level of that voxel at the point of the simulation divided by the maximum respective value of all voxels. The value is normalized by the current maximum average pressure.

Other rendering options such as the colour scale, voxel scaling, shape, and aggregation level across each axis are also available to the user, as part of iCrowd's unified voxel grid rendering mechanism.

Pressure levels are also visualized by changing the colour of each agent according to the pressure they are currently experiencing. A colour-scale ranging from yellow (which is the default colour of an agent) to red (indicating the maximum pressure – 200N) is used, as shown in Figure 10.

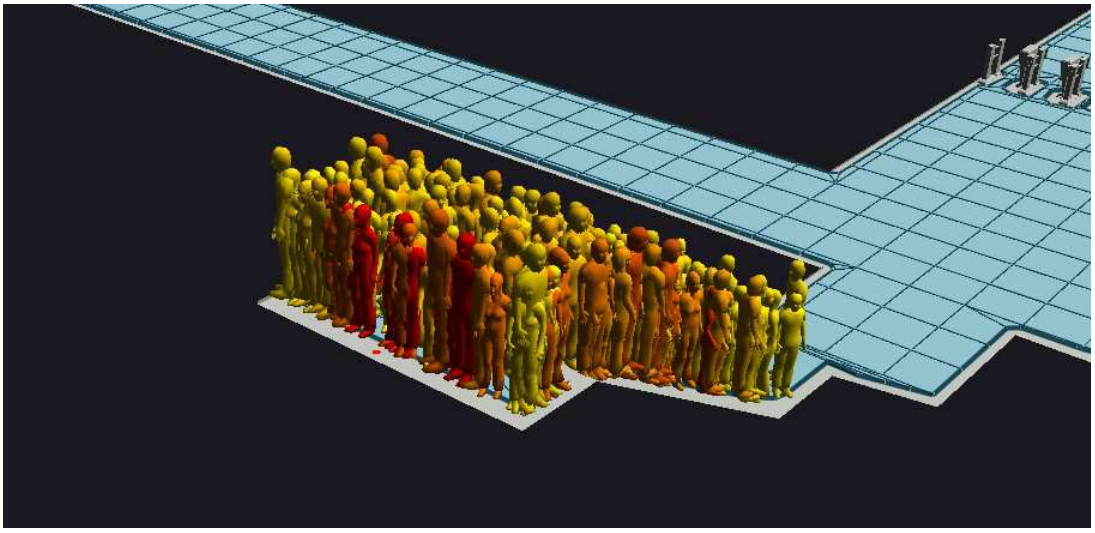

Figure 10: Pressure levels of individual agents in a stampede (yellow means no pressure, red means maximum pressure)

3.4.5.3 Monitored Areas

The **Detection Module** implements a realistic field of view for cameras and guards, as described in section 3.4.2. At each step of the simulation, each detector has a subset of the environment within its field of view, which may include entities such as agents or other objects, and polygons of the navigation mesh. A rendering function was added to the module to provide a real-time visualization of the visible entities and areas.

During the simulation, entities that are visible by at least one detector are marked by a green bounding box around them. Upon selecting a detector, the user can choose to render a bounding box around only the entities that are visible by this specific detector. This is shown in Figure 6.

In order to inspect the overall coverage of the environment by detectors, the Detection Module was also extended to keep its own sparse voxel grid covering only the walkable surfaces of the environment. Each voxel contains a history of boolean values indicating whether the voxel was visible by any detector at each point of the simulation, along with the total time for which the voxel has been visible. The detection voxel grid enables the rendering of a colour-scaled heatmap-type visualization of coverage of the 3D model. The user can render the following information:

- **Overall coverage**: The value of each voxel is the time for which it has been visible divided by the total time that the detection module has been sampling the data, i.e. the total time of the simulation. The value is normalized by the total duration of the simulation.
- **Overall relative coverage**: The value of each voxel is the time for which it has been visible divided by the maximum time that any voxel has been visible. The difference with the "Overall" coverage above is that this value is normalized by the maximum time that any voxel has been visible, instead of using the total duration of the simulation.

  Note: If there are static cameras that have been enabled at all steps of the simulation, then the voxels that are visible by them will have been visible for the entire duration of the simulation. Their "monitored" time will be the maximum time by which all times will be divided. As such, this

*representation will be the same as **Overall** as the values will be normalized by the same time, equal to the duration of the simulation.*

- **Coverage at a specific time step**: Here, the user selects an earlier time step of the simulation and sees a heatmap of the areas that were covered at that time. The value of each voxel is either 1 or 0, indicating that, at the selected time step, the area was either visible or not visible by at least one detector. A time scaled playback function is available showing how cameras and guards moved through the environment.

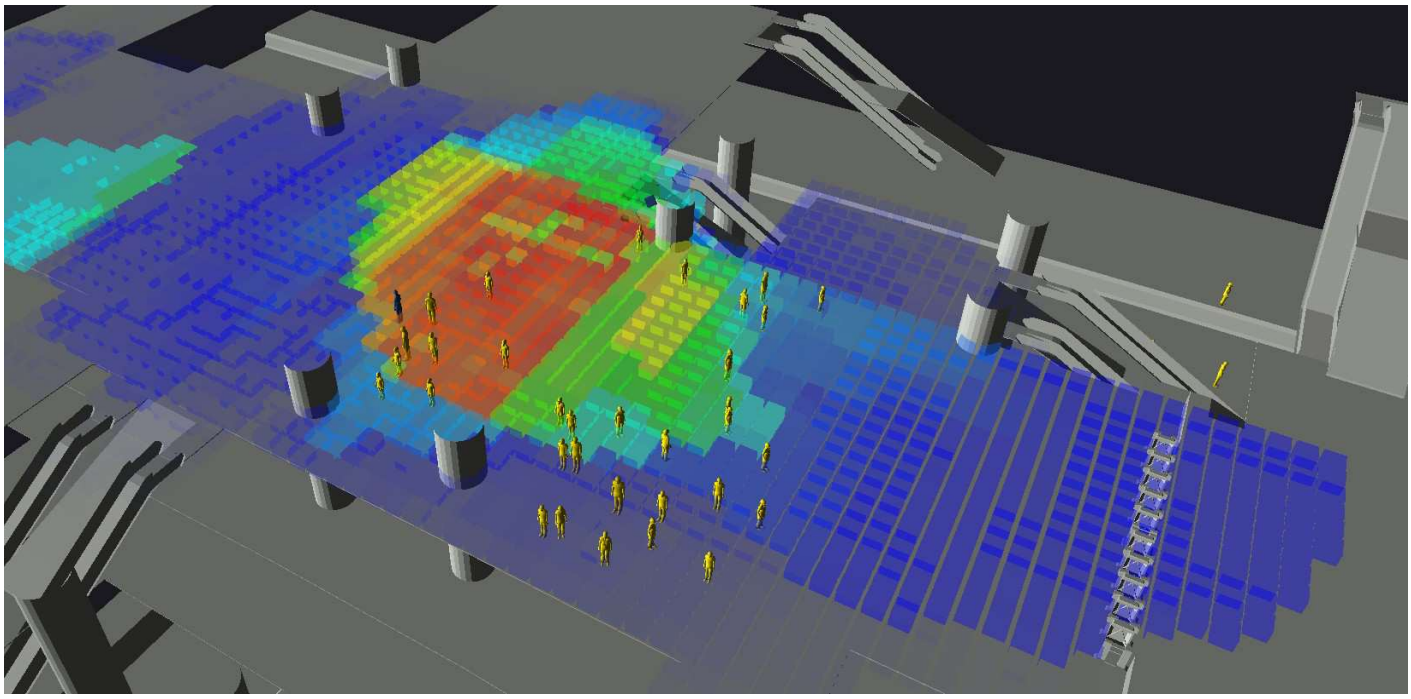

Figure 11: Heatmap of detectors' coverage by guards and cameras

Other rendering options such as the colour scale, voxel scaling, shape, and aggregation level across each axis are also available to the user, as part of iCrowd's unified voxel grid rendering mechanism. A sample representation of the relative overall coverage is shown in Figure 11.

3.4.6 Integration with DMS

The new **DMS Module** has been developed within iCrowd to provide an integration with DMS; an **Apache Kafka** distributed messaging system. This module implements API calls through HTTPS to Kafka to log in,

Figure 12: A GUI window allowing the user to send custom messages to the DMS server

subscribe to topics, send messages to topics, and poll for messages from topics. The implementation of the module is generic, i.e. it does not include any SAFETY4RAILS-related calls, topic names, responses, etc.

Rather, this information is expected to be provided at runtime by any module that wishes to communicate with DMS.

For demonstration and debugging purposes, the DMS Module provides the user with a simple GUI to construct and send messages manually to any topic, as well as to read any polled messages. This GUI is shown in Figure 13 and Figure 12.

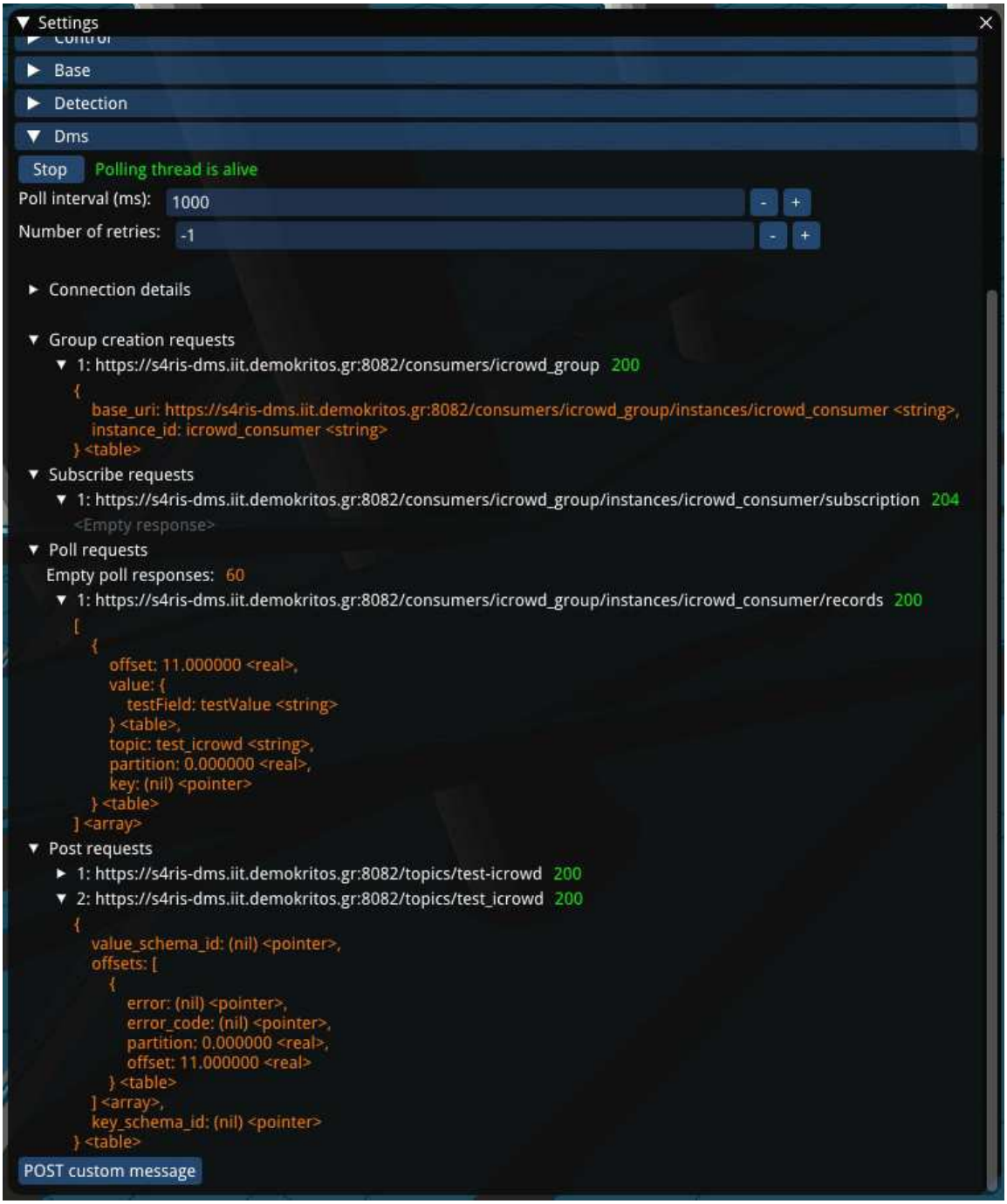

Figure 13: The DMS module showing the result of all HTTPS calls made to the DMS server

In the context of SAFETY4RAILS, the **S4R Module** has been developed which implements all functionalities related to the project. Upon initialization, the user sets up the DMS Module with a URL, username, and password. Then, the S4R Module adds the topics it wishes to poll for messages, such as the topic on which BB3D posts the results of its simulations. When an event happens in the simulation, such as the detonation of a bomb, the S4R module constructs a JSON message and passes it onto the DMS module to send it to a specific topic. These interactions with the BB3D tool are described in detail in section 4.1.3.

3.5 Training

The iCrowd simulator aims to serve as an intuitive and helpful assessment tool to enable resilience capabilities primarily for the phases identification and protection (combined as “prevention” in the MDM and EGO simulation exercises). The iCrowd simulator can be utilized as an innovative tool for training purposes in a gamified environment for the railway operators, aiming to provide an intuitive environment that can be used in the development of and evaluation of railway safety and security design sessions. As a training tool, iCrowd can help the security operators evolve or sharpen their strategic decision skills and train their understanding regarding the related resilience strategies.

The **iCrowd Training Platform** follows a **Simulation-as-a-Service** model [19] that has been set up by NCSRD to provide the iCrowd simulator to end-users or other interested parties as an online service. This was implemented and tested in the context of the previous EU-funded project TRESSPASS, where, after following the training process below, the end-users were able to use the iCrowd simulator completely on their own. Specifically, an isolated, secure, Linux-based, remotely controlled virtual machine (VM) is provided to each interested party, which is preloaded with the iCrowd simulator and its dependencies. The user requests a time slot from NCSRD, and is then provided with a URL and a password that can be used to access their VM from any web browser. Any modern consumer-grade computer can be used, since no specialized software or hardware is required from the user’s side other than a modern web browser and a stable internet connection.

The training procedure is comprised of the following phases:

1. **Initialization phase**: The simulation scenario’s core aspects are defined.

   End-users provide detailed scenarios in the same way they did for their physical pilots, the behaviours of the malicious actors are strictly defined. The options regarding the freedom of movement of the crowd are examined and defined.

2. **Introductory phase**: End-users are introduced to the iCrowd simulator and its initialization mechanism by NCSRD.

   The hands-on introduction session is done through a teleconference, while the trainees have access to isolated, remotely controlled, Linux-based virtual machines (VMs) that NCSRD sets up on a high performance server, so that they can follow along with the trainer’s instructions. Initial configurations are collected, including initial conditions, camera configurations, and any other parameters are given by the end-users.

3. **Simulation phase**: End-users execute their scenarios with initial configurations and retrieve the results.

   The end-users can execute their scenarios, adjust them and run them again, and extract the results in CSV files. The end-user as the operator of the simulator can observe the execution of the simulation in real time, get metrics of the desired indicators, and pause, resume, or make the simulation go faster than in real-time. The operator can also inspect agents, cameras, and checkpoints, by simply clicking on them and navigating through their settings window. Checkpoints and cameras can be selectively enabled and disabled, to facilitate what-if scenarios.

4. **Result phase**: End-users review and evaluate the results. If necessary, the configuration parameters are adjusted and the procedure returns to the Simulation phase until satisfying results are obtained.

   The purpose of the results phase is for the end-users to sharpen their decision-making skills by analysing the outputs and trying different strategies with similar or different configurations. If needed, the end-users request more time with the simulator to run further simulations and collect new results. Finally, the executions’ results are reviewed and conclusions were drawn by experts.

4 Integration with Other Tools

As already described in chapter 3, the iCrowd simulator features a network communication system that enables it to communicate with external tools using a flexible JSON protocol. The simulator also features connectivity through HTTP, along with a dedicated implementation for connecting to an **Apache Kafka server**, such as SAFETY4RAILS's DMS. This enables iCrowd to integrate with other tools of the project, to provide a more informed and functionality-rich simulation platform, along with more useful metrics for the end-users to effectively evaluate their resilience strategies. Other tools can also receive information from iCrowd, such as movement trajectories, status updates about agents or assets, or results. Since DMS retains messages instead of just passing them through, it is possible to have both synchronous (real-time) and asynchronous integrations.

In the context of Task 5.2, the iCrowd simulator has been integrated only with RINA's BB3D tool and provides information that is relevant only to that. The integration with BB3D is explained in detail in the next section.

4.1 BB3D

To provide users with a numerical means to approach blast design consequent to a deliberate bomb attack, an in-house predictive tool based on the literature empirical data was developed by RINA (formerly D'Appolonia) in the framework of SECURESTATION (CORDIS | European Commission (europa.eu)), a research co-funded project of the Seventh Framework Programme of the European Commission. Such tool, referred to as BomBlast3d [16], computes the loading due to the blast wave impact over structures such as buildings, and supplies the main physical quantities of interest both over the wall surface of three-dimensional models, virtually reproducing potential attractive targets for terrorists, and in air. These results can be visualised and used to support blast analysts' assessments and decision makers.

The principal rationale that drove its development is based on the following principles:

1. Minimal set of input data to ease its utilization
2. Fast and stable computing to guarantee the achievement of results in short time
3. Use of open and standard files to limit commercial software restrictions

Considering the rationale for development, BB3D is written in FORTRAN programming language and works on Windows operating system. It uses open-source (ASCII) files as input (STL) and output (VTK), which can be visualised through the open-source visualization tool Paraview 14 [20]. The computing is fast and intrinsically stable because it uses data referring to hemispherical surface bursts reported in the Unified Facilities Criteria (UFC) manual (Department of Defence).

Considering the capabilities described, the use of the proposed novel methodology is desirable in the preparatory and retrofit design in view of evaluating blast-induced adverse consequences and, accordingly, to identify and effectively set up the most appropriate protective strategies and countermeasures.

4.1.1 Purpose of Integration

BB3D was integrated with iCrowd in the context of the MDM simulation. The goal was for iCrowd to realistically simulate the impact of a bomb explosion on the crowd and infrastructure, regarding injuries, fatalities, and disabled assets of the environment. BB3D provides an on-demand simulation platform that accepts a request for a bomb explosion, simulates it, and returns the results. The request includes the details of the bomb, its location in the predefined 3D geometry, and a summary of the crowd distribution around it. The results include a 3D voxel grid containing the pressure levels due to the explosion and survival probability of a human at each point in space, and a list of damage levels for each part of the infrastructure. The detailed communication protocol is described in section 4.1.3.

As a result of this integration, a user of the iCrowd simulator can start a simulation scenario and dynamically define a place and time for a bomb to be detonated. They can even define multiple bombs, to be detonated at various times. Then, when the simulation reaches that point in time, it will be automatically paused, the request for the bomb explosion simulation will be automatically sent to BB3D through DMS, and when BB3D responds with the results of the simulation, they will be retrieved, applied, and the simulation will automatically continue. Based on the survival probabilities and infrastructure damages sent by BB3D, iCrowd will automatically apply injury levels to nearby agents, and disable any relevant active assets.

4.1.2 Visualization

The results of a bomb explosion as calculated by BB3D can be visualized within iCrowd without the need for external software. The results include a 3D voxel grid, where each voxel represents a rectangular box at a specific position in space, and contains the level of pressure and survival probability due to the explosion for that area. This information can be visualized by iCrowd using its unified voxel grid visualization mechanism (as described in Native Visualization of Results), that allows the user to choose a colour-scale among other options to efficiently visualize each aspect of the results.

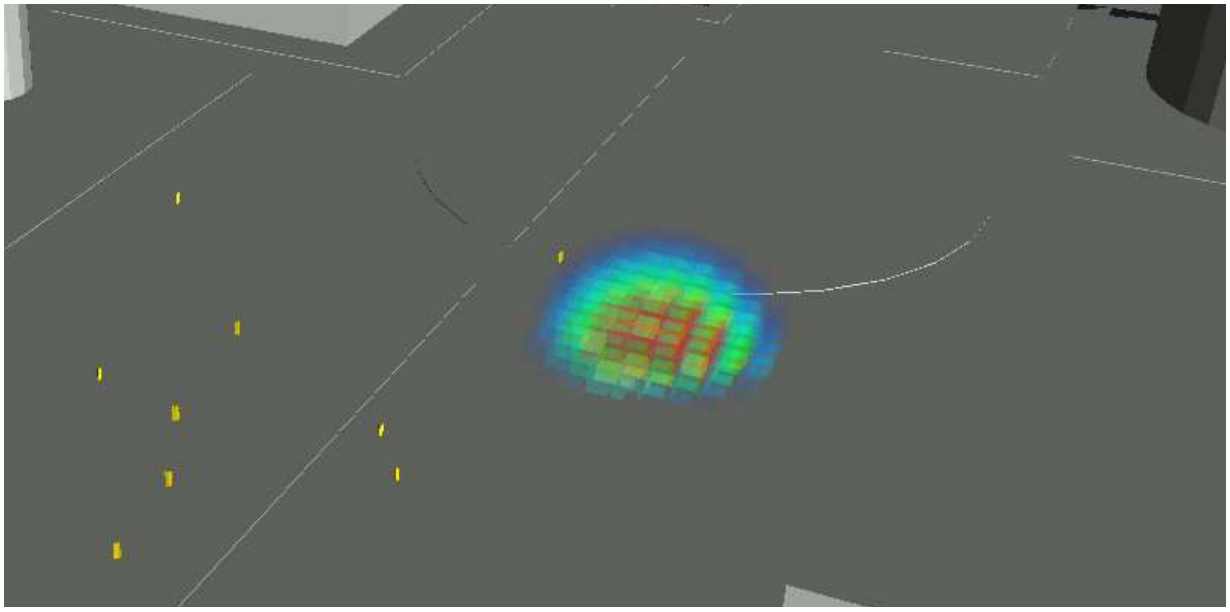

Figure 14: Visualization of BB3D's results of a bomb explosion

4.1.3 Communication Protocol (API)

To communicate the results of the bomb explosion simulator, BB3D uses sampling across all 3 axes of the area around the detonation point to create a 3D voxel grid. Each element of the grid corresponds to a voxel; a cube with a predefined size, that contains information about the area it covers, such as the average pressure and survival probability. A visualization of the voxel grid can be seen in Figure 14.

The voxel configuration is done on a per model basis, but the only differences are the size of the grid and the size of the voxels, since the rest of the parameters remain the same. To accommodate the communication of the grid (or parts of it), the grid has been linearized by exploiting the same idea that is used in many programming languages to map a multi-dimensional array to a one-dimensional array for storing it in memory. By defining an order of the dimensions (e.g. first X, then Y, then Z), a scan order for each dimension according to its coordinates (e.g. X low-to-high, Y low-to-high, Z high-to-low), and the number of elements for each dimension (which correspond to the dimensions of the grid), we can express the 3-dimensional voxel grid as a 1-dimensional array of voxels. Each voxel in the 1-dimensional array can be identified by a single integer value (index) instead of the 3 indices that would be required in the 3D representation. With the voxel grid as a 1D array, the required information can be easily communicated between the simulators.

Due to time constraints, the integration with the BB3D tool was done assuming a predetermined 3D model. As such, the model itself is not part of the communication. However, since both iCrowd and BB3D can easily be adjusted to use different models, for future integrations it would be beneficial to have the model be a part of the request message (see below), either as a whole, or as a reference to a shared database of 3D models.

4.1.3.1 Request Message (iCrowd to BB3D)

At the time step of a bomb's detonation, iCrowd prepares and sends a message to BB3D by posting it on its topic on DMS. The message contains information about the simulation's state (current timestamp), the bomb(s) (position, heading, type, other parameters), and the crowd's positions.

Most of the information is quite straightforward and does not need any special formatting, except for the crowd's positions. Since the simulations were expected to run with thousands of agents, it would be inefficient to create and transfer a message containing the positions of all agents, due to its large size. As such, a predefined radius is used to filter the agents whose positions are contained in the message.

Overall, the message that is sent by iCrowd to BB3D via DMS has the following format:

```
{
    “ts”: 11.97,
    "bombs": [                          # List of bombs
        {
            "name": "bomb0",            # Name of bomb (for reference purposes)
            "position": [               # Position of bomb in the model space
                -123.45, 0.0, 123.45
            ],
            "heading": [                # Heading vector of bomb
                1.0, 0.0, 0.0
            ],
            …                           # Other parameters related to the bomb
        },
        ...
    ],
    "crowdSize": 123
    "crowd": [
        [ -123.45, 0.0, 123.45 ],
        [ -123.45, 0.0, 123.67 ],
        ...
    ]
}
```

4.1.3.2 Response Message (BB3D to iCrowd)

The information of a single voxel can be serialized as follows:

```
{
    "peakIncP": 123.45,
    "specIncImp": 123.45,
    "survProb": 100.0
}
```

Then, a list of all voxels can be serialized as a JSON array. Since all voxels need to be sent, an array containing all voxels, starting from index 0 and going all the way up to the last, is created and sent, like the one below.

```
"voxelsDense": [
    { Information about voxel 0 }
    { Information about voxel 1 }
    ...
    { Information about voxel N-1 }
]
```

Since the area that the voxel grid needs to cover depends on the outcome of the explosion, it is decided by the BB3D simulator. For the grid to be parsed and decoded correctly by iCrowd, the response message needs to contain the grid's configuration details. A voxel grid is defined by its width, height, depth, origin point, and cell size. The voxel configuration is sent using the following format:

```
"voxelGridConfig": {
    "width": 112,    # The number of voxels across the X axis of the grid
    "height": 13,    # The number of voxels across the Y axis of the grid
    "depth": 80,     # The number of voxels across the Z axis of the grid
    "cellSize": [ 10.0, 10.0, 10.0 ],    # The size of each voxel/cell

    # The position of the first voxel (with index 0)
    "origin": [ -559.15, 0.0, -521.64 ]
}
```

To transfer the damage results of the infrastructure, a different approach is used. BB3D samples the environment using the triangles of the geometry, and creates a JSON element for each one. An array of these elements is transferred to iCrowd, using the following format:

```
“solidWallResults”: [
    {
        "centrCoords": [
            -83.06,
            21.45,
            0.75
        ],
        ... (Infomration about this triangle)
    },
    { Next triangle },
    { Next triangle }
    ...
]
```

Overall, the message that is sent by BB3D to iCrowd via DMS has the following format:

```
{
    “ts”: 19.97,
    “nextTs”: -1.0,
    “results”: [
        {
            “bombName”: “bomb1”,
            “bombCoords”: [
                -123.45, 0.0, 123.45
            ]
            ... (Bomb information fields),
            “voxelGridConfig”: {
                "width": 10,
                "depth": 10,
                "height": 7,
                "cellSize": [
                    1.75,
                    1.75,
                    1
                ],
                "origin": [
                    -148.74,
                    11.26,
                    0.1
                ]
            },
            “voxelsDense”: [
                { Information about voxel 0 },
                { Information about voxel 1 },
                ...
                { Information about voxel N-1 },
            ]
            “solidWallResults”: [
                { Information about a triangle },
                { Information about another triangle },
                ...
            ]
        }
    ]
}
```

5 Summary and Conclusion

In this deliverable report we described the use of the iCrowd simulator in the context of SAFETY4RAILS and the extensions that were developed as part of Task 5.2.

The report has described the iCrowd simulator on which the simulation platform was built, its modules that were developed in the context of this project, and its potential use as a training platform. Furthermore, this deliverable sets out the integration of iCrowd with other tools, along with the potential of integration with more tools.

Overall, the iCrowd simulator was extended to provide object detection and detection evasion functionalities, to properly simulate automatic escalators, and to calculate pressure levels among the agents, in order to incorporate the KPIs that were proposed by NCSRD and the respective end-users. Visualization mechanisms were also developed into iCrowd to provide a user-friendly visualization of the results in real-time without the need for specialized external visualization software.

The above functionality was evaluated by 2 scenarios (the MDM and EGO simulation exercises). These mainly focused on crowd evacuation, breach of a restricted area by malicious actors, and a bomb explosion which was simulated by the BB3D tool. Since this deliverable D5.2 is a public report, details about the scenarios that were simulated and their inputs and outputs are presented only in deliverable D5.7 which is confidential.

5.1 Future work

While the iCrowd simulator was extended to adequately cover the MDM and EGO simulation exercises, there is still the potential for a more intelligent, robust, and overall useful simulation platform. Some of the requirements were implemented but not tested in the MDM or EGO simulation exercises due to time constraints prohibiting the adjustment of those exercises. These were:

- **iCrowd_03 - Simulate crowd behaviour considering cyber-agents**: Implement a behavioural model for human agents based on the default (from input), which dynamically adapts to new knowledge. This can happen when the agent has a cyber agent in their field of view which presents new information (such as an electronic board), or the agent gets in close proximity to another agent who has already acquired the new information.
- **iCrowd_06 - Guards' distraction simulation**: Temporarily deactivate the guard's ability to report an incident and/or act on it, affecting the rest of the simulation and any interfacing with other real systems. This can be triggered by time to simulate a cyber attack (assuming the guard receives a call, noise in their headset, etc.), by interacting with a different agent to simulate a physical attack (assuming a co conspirator distracts the guard by talking to them), or periodically or based on real-time metrics to simulate the guard's psychological state (tired, anxious, day-dreaming).

Other possible extensions of the iCrowd simulator include behaviours and integrations with external software:

- **Intelligent complex behaviours for agents**: A sophisticated behaviour can be modelled as a behaviour-tree which will describe the behaviour of the agents. This behaviour can incorporate knowledge propagation (requirement iCrowd_03), advanced decision making during an attack, and a more detailed movement logic within a locked environment (such as the locked station in the MDM scenario) or a dark environment (such as the cyber-attack in the EGO scenario).
- **Explicit behaviours for malicious agents**: Make malicious actors in the simulation follow a more complex behaviour tree, including a more intelligent evasion of cameras and guards by waiting for a guard to pass instead of continuously walking around them, deliberately mixing into the crowd by targeting areas with high congestion levels.
- **Integrate with CaESAR**: Complete the integration that was planned with CaESAR to provide realistic arrival rates, train schedules, and take into account any externally interconnected infrastructures.
- **Tracking cameras and anomaly detection for suspicious behaviours**: Use the movement trajectories of agents, both malicious and non-malicious, to train an anomaly detection system to recognize suspicious behaviours. Use the simulated cameras to remove parts of trajectories of agents while they are not visible by any, to provide realistic data as they would come from a real tracking

system. NCSRD has already developed such as a system, that can be extended and improved using this functionality [21]–[23].

Bibliography

[1] V. Kountouriotis, S. C. Thomopoulos, and Y. Papelis, “An agent-based crowd behaviour model for real time crowd behaviour simulation,” *Pattern Recognition Letters*, vol. 44, pp. 30–38, 2014.

[2] S. Daveas and S. C. Thomopoulos, “Embedding a distributed simulator in a fully-operational control and command airport security system,” in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII*, 2018, vol. 10646, p. 106461C.

[3] V. I. Kountouriotis, M. Paterakis, and S. C. Thomopoulos, “iCrowd: agent-based behavior modeling and crowd simulator,” in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, 2016, vol. 9842, p. 98420Q.

[4] S. C. Thomopoulos, C. Kyriakopoulos, and K. Panou, “Impact assessment and mitigation strategies in rail/metro infrastructure with the use of iCrowd Simulator,” Orlando, Florida United States, 2022. [Online]. Available: http://www.spie.org/si220

[5] C. Di Chio, *Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings*. Springer Science & Business Media, 2011.

[6] C. Di Chio *et al.*, “Applications of Evolutionary Computation-EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings: Preface,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6024, no. PART 1, p. VII–XI, 2010.

[7] D. Isla, “GDC 2005 proceeding: Handling complexity in the halo 2 AI,” *Retrieved October*, vol. 21, p. 2009, 2005.

[8] M. Colledanchise and P. Ögren, “How behavior trees modularize robustness and safety in hybrid systems,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1482–1488.

[9] S. J. Guy *et al.*, “Clearpath: highly parallel collision avoidance for multi-agent simulation,” in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 177–187.

[10] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” *The international journal of robotics research*, vol. 17, no. 7, pp. 760–772, 1998.

[11] D. Helbing and P. Molnar, “Social force model for pedestrian dynamics,” *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[12] D. Helbing, I. Farkas, and T. Vicsek, “Simulating dynamical features of escape panic,” *Nature*, vol. 407, no. 6803, pp. 487–490, 2000.

[13] “Apache Kafka - An open-source distributed event streaming platform.” [Online]. Available: https://kafka.apache.org/

[14] “Lua: a powerful, efficient, lightweight, embeddable scripting language.” [Online]. Available: https://www.lua.org/about.html

[15] “Wavefront .obj file,” *Wikipedia*. Mar. 28, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Wavefront_.obj_file&oldid=1079833682

[16] E. Costa, “Implementation of an empirical tool for fast prediction of bomb airblast loading,” *International Journal of Protective Structures*, vol. 10, no. 1, pp. 54–72, 2019.

[17] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[18] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics research*, Springer, 2011, pp. 3–19.

[19] C. Kyriakopoulos and S. C. Thomopoulos, “iCrowd Simulation-as-a-Service (SaaS): a distributed and remotely accessible simulation platform,” Orlando, Florida United States, 2022. [Online]. Available: http://www.spie.org/si220

[20] *ParaView - An open-source multiple-platform application for interactive, scientific visualization*. [Online]. Available: https://www.paraview.org/

[21] S. C. Thomopoulos and C. Kyriakopoulos, “Anomaly detection with noisy and missing data using a deep learning architecture,” in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXX*, 2021, vol. 11756, p. 117560R.

[22] G. Bouritsas, S. Daveas, A. Danelakis, and S. C. Thomopoulos, “Automated real-time anomaly detection in human trajectories using sequence to sequence networks,” in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2019, pp. 1–8.

[23] S. C. Thomopoulos, S. Daveas, and A. Danelakis, “Automated real-time risk assessment for airport passengers using a deep learning architecture,” in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVIII*, 2019, vol. 11018, p. 110180O.

ANNEX I. LIST OF ABBREVIATIONS

Term	Definition/description
API	Application Programming Interface
BB3D	Bomb-Blast 3D
CCTV	Closed-Circuit TeleVision
CSV	Comma-Separated Values
DMS	Distributed Messaging System
EER	Electronics Equipment Room
ETS	Emergency Trip System
FIFO	First-In-First-Out
GUI	Graphical User Interface
HTTP(S)	HyperText Transfer Protocol (Secure)
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
PCR	Potentially Colliding Region
TCP	Transmission Control Protocol
UC	Use-Case
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine

Table 3: List of abbreviations

SAFETY4RAILS

Partners:

Fraunhofer EMI
UIC
Metro de Madrid
EGO Genel Müdürlüğü
RFI Rete Ferroviaria Italiana
Gruppo Ferrovie dello Stato Italiane
TCDD
Leonardo
ceis avisa partners
STAM Mastering Excellence
IC information company
RMIT University
University of Reading
etra I+D
Demokritos National Centre for Scientific Research
RINA
WINGS ICT Solutions
Newcastle University
EOS European Organisation for Security
CybeServices Made in Europe
erGE
Innova Integra
LAUREA Ammattikorkeakoulu University of Applied Sciences
FGC Ferrocarrils de la Generalitat de Catalunya
MTRS
INTRACOM TELECOM
Universitas Miguel Hernández
TREE Technology
Elbit Systems C4I and Cyber
ProRail
Comune di Milano

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 883532.