

***Advanced Virtual Reality Framework for Studying Outdoor Fire Evacuation Dynamics***

December 2024

**Namrata Bhaumik, MACS**

**Oleksandr Rossolov, PhD**

**Ahsan Habib, PhD**

Dalhousie Transportation Collaboratory (DalTRAC)

School of Planning, and Department of Civil and Resource Engineering Room #B105, Dalhousie University, PO Box: 15000

1360 Barrington Street, Halifax, NS Canada, B3H 4R2

**Contents**

[1. Research framework 2](#_Toc184821434)

[*1.1* *3D game environment* 5](#_Toc184821435)

[*1.2 Static objects* 5](#_Toc184821436)

[1.2.1 Tree 6](#_Toc184821437)

[1.2.2 Bus 6](#_Toc184821438)

[1.3 Dynamic objects 7](#_Toc184821439)

[1.3.1 Pedestrians 7](#_Toc184821440)

[1.3.2 Fire 12](#_Toc184821441)

[2. Evacuation scenarios development 16](#_Toc184821442)

[3. Virtual reality evacuation experiment 19](#_Toc184821443)

[4. Collected analytics 21](#_Toc184821444)

[ANNEX 1 – Questionnaire 23](#_Toc184821445)

# Research framework

**Overview**:

This project utilizes Virtual Reality (VR) and simulation software to analyze pedestrian behavior during fire evacuation scenarios. The main objective is to create a virtual environment using Spring Garden Road, Halifax, NS, Canada where people can observe and interact with pedestrians as they react to an emergency, such as a fire. The project aims to create a realistic VR simulation where pedestrians react to an emergency evacuation during a fire. This involves simulating a variety of pedestrians, each with different behaviors and speeds, moving through the cityscape toward designated exit (marshal) points. The simulation includes various pedestrian interactions, such as avoiding other pedestrians, following paths, and reacting to obstacles (e.g., other pedestrians) and the presence of the player in the environment. Pedestrians are randomly assigned different routes and speeds, and they are programmed to avoid collisions with both the player and other pedestrians. The evacuation procedure is explored within the scenario-based games. By experiencing these scenarios in VR, users can gain insights into potential evacuation bottlenecks and improve safety protocols.

**Purpose of the Evacuation Simulation:**

The primary goal of the evacuation project is to observe and analyze how pedestrians react in emergency evacuation situations, particularly during fires. The VR environment offers a safe, controlled way to study human behavior without risking real-life harm. The key purposes include:

1. **Behavioral Analysis:** The simulation helps researchers, city planners, and emergency management personnel understand how people behave during a fire evacuation, including how they navigate obstacles, make decisions, and avoid collisions with other pedestrians.
2. **Evacuation Efficiency:** By studying pedestrian flow and movement patterns, the project aims to identify potential bottlenecks or delays in the evacuation process, helping to optimize exit points, pathways, and crowd management strategies.
3. **Safety Improvements:** By simulating various fire and evacuation scenarios, the project provides valuable data that can be used to design safer buildings, improve evacuation routes, and create more efficient emergency protocols. Understanding how different groups of people react can help tailor evacuation procedures to ensure their safety.
4. **Training and Education:** The VR simulation can be used for training purposes, allowing users (e.g., emergency responders, city planners, or safety managers) to practice evacuation strategies in a realistic virtual environment. This training could be particularly useful in situations where traditional drills are not feasible.

By observing pedestrians' actions and interactions in the simulation, the project aims to improve our understanding of crowd dynamics, which can ultimately lead to better-prepared cities and more efficient fire evacuation plans.

**Software and Technologies Used:**

1. **Unity 3D (2022.3.46f1):** Unity is the primary game engine used for creating the VR simulation. It provides tools for building 3D environments, physics simulations, and scripting interactions. Unity's XR (Extended Reality) packages enable VR integration, which allows users to experience the evacuation scenario in an immersive virtual environment.
2. **Oculus Meta Quest 3:** The project leverages Meta’s Oculus Quest 3 as the VR hardware to provide an interactive experience. The XR (VR) setup ensures that the user can navigate the virtual cityscape, interact with pedestrians, and experience the dynamics of fire evacuation firsthand.
3. **VS Code Editor:** C# is the scripting language used to implement the logic for pedestrian movement, collision avoidance, and simulation of evacuation behavior. Scripts manage the behaviors of both the player and pedestrians, controlling movement, interactions, and reactions based on the environment.
4. **SketchUp (for City Models):** SketchUp is used for creating 3D models of Downtown Halifax. Buildings, streets, sidewalks, and other components are imported into Unity to make it realistic. This city serves as the backdrop for the pedestrian evacuation simulation.
5. **Audio Tools (for sound effects):** Audio sources are downloaded for creating sound of fire and human voice is recorded for safety guidance during evacuation. These sound cues play a crucial role in simulating an emergency evacuation scenario.

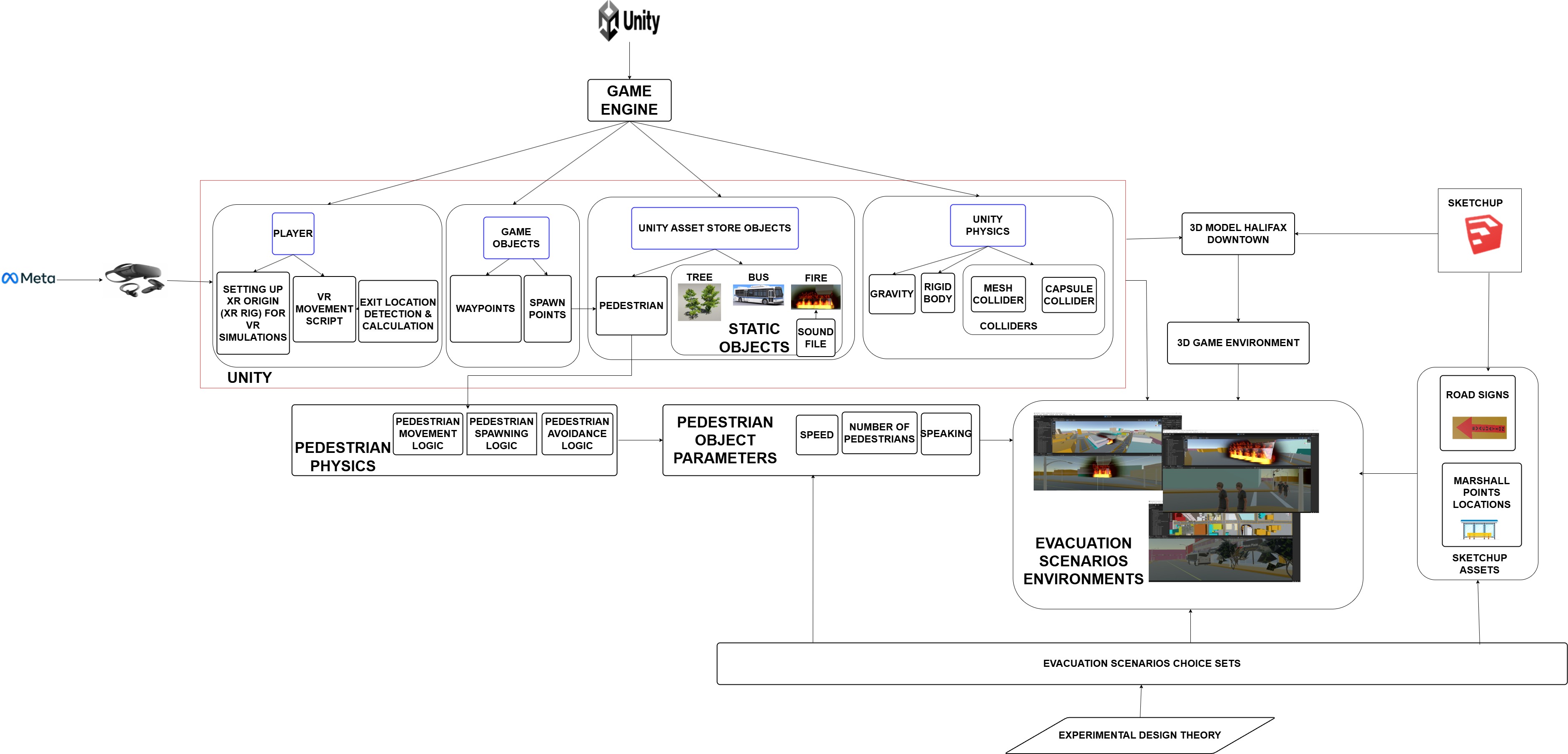


Figure 1. The flow chart of the research framework

## *3D game environment*

**Figure 2** represents the study area implemented in SketchUp.

A map of a city

Description automatically generated

Figure 2. – Case study area of Halifax city

The model has been enhanced with the static objects, such as: trees and buses located on the evacuation marshal points and also the dynamic objects such as pedestrians and fire asset.

**Preparing the SketchUp Model for Unity**

To ensure the SketchUp city model integrates seamlessly with Unity's physics and gameplay mechanics, a 3D Plane object with a Mesh Collider was added. This plane serves as a foundational layer to prevent both the player and pedestrian objects from falling through the environment. The mesh collider provides a collision surface that aligns with the model, allowing objects to interact realistically with the terrain. This adjustment ensures stability during gameplay and enables smooth navigation for all dynamic objects in the scene.

## *1.2 Static objects*

In Unity, static assets refer to objects that do not move or change dynamically during runtime. These are typically used to create the environment, provide structure, and populate the scene. Static assets help to build the foundation of the scene in Unity, providing a sense of place and structure without altering during gameplay. They are also optimized for performance, as Unity can batch and process static objects more efficiently.

Static assets include:

* Meshes: 3D models like buildings, trees, or furniture that do not move or animate.
* Textures: Visual elements applied to meshes to give them surface detail (e.g., brick, wood, metal).
* Materials: Define the appearance of objects, such as their color, glossiness, or roughness.
* Lights: Non-moving light sources that illuminate the scene.
* Audio: Sound elements that do not change or interact dynamically, such as background music or ambient noise.

Next chapter discusses the logic of their location considering the experimental design.

The static objects used are the following:

### 1.2.1 Tree

Positioned as a stationary object within the scenario at the marshal points during the evacuation (Fig. 3).

A couple of trees with shadows

Description automatically generated

Figure 3. – Unity's Tree Asset

### 1.2.2 Bus

Positioned as a stationary object within the scenario at the marshal points during the evacuation (Fig. 4).

A white bus with blue stripe

Description automatically generated

Figure *4. –* Unity's Bus Asset

## 1.3 Dynamic objects

In Unity, dynamic assets refer to objects or elements in the scene that can change or move during runtime, typically interacting with player actions, other objects, or game mechanics. These assets are essential for creating interactive and immersive gameplay experiences. Dynamic assets are integral to gameplay interactivity, allowing for player-driven changes in the environment, characters, and other game systems. They are usually processed in real-time, making them more resource-intensive than static assets.

Dynamic assets characteristics include:

* GameObjects: Any object that can move, rotate, or change properties during gameplay, like characters or vehicles.
* Animations: Predefined movements or behaviors for objects, often applied to characters or interactive elements.
* Rigidbody: A component attached to objects that enables them to be affected by physics, allowing for movement, collisions, and realistic interactions.

**Figure 6** represents the pedestrian movement logic, **Figure 7** – the pedestrian cloning logic**, Figure 8** – VR Player movement dynamics, and **Figure 9** – the audio file addition to the player object.

### 1.3.1 Pedestrians

The dynamic asset used in the project is the pedestrian asset. These are characters that move throughout the scene, following specific paths or waypoints.

The pedestrian asset prefab is configured with the **Animation Type Humanoid** in Unity. This setting ensures that the characters move and walk with human-like motions, adding to the realism of the simulation. Their movement is controlled by scripts that allow for behaviors such as random direction changes, speed variations, and path following, making them behave like realistic pedestrians in the virtual city.

**Collision Avoidance**

Collision avoidance is a fundamental aspect of the VR evacuation simulation, ensuring smooth and realistic interactions between pedestrians and the player as they navigate the virtual environment. This system is built upon Unity's robust physics framework, combining CharacterControllers, Colliders, and, where necessary, Rigidbody components to achieve precise and efficient navigation.

The **CharacterController** plays a pivotal role in controlling both the player and pedestrians. Unlike Rigidbody, which relies on physics-based forces, the CharacterController provides lightweight, non-physics-based movement, making it ideal for precise and predictable navigation. It inherently supports collision detection, allowing characters to sense obstacles and adjust their path dynamically. For instance, when a pedestrian detects a collision with another pedestrian or the player, the CharacterController halts the movement temporarily, and the script recalculates a new path or adjusts the direction to avoid overlap. This approach ensures smooth, realistic responses without the complexity of excessive physics computations.

**Colliders** are used extensively to define the physical shapes of objects for collision detection. For humanoid characters like pedestrians, Capsule Colliders are used to encapsulate their body shape, while Box Colliders are usually applied to static environmental objects such as buildings and obstacles. The player’s XR Rig, too, has a Collider attached, which allows pedestrians to detect the player’s presence and modify their movement accordingly. When a pedestrian encounters the player’s Collider, the system triggers a behavior adjustment, often randomized to simulate realistic decision-making, such as choosing whether to move left or right.

While **Rigidbody** components are generally used to apply real-world physics to objects, their use in pedestrian characters is limited to maintaining control and stability. For precise motion, Rigidbodies on pedestrians are set to kinematic or disabled entirely. This avoids unintended physics interactions that could disrupt controlled movement. However, the player’s XR Rig employs a Rigidbody to maintain stability on uneven surfaces and prevent falling through the ground, especially in scenarios where gravity or other forces might come into play.

Player and pedestrian interactions are seamlessly integrated into this system. The player’s XR Rig is treated as a dynamic obstacle, prompting pedestrians to detect its Collider and respond accordingly. Similarly, pedestrians detect each other’s presence and dynamically adjust their paths to avoid congestion, maintaining a fluid flow of movement. This interaction design ensures that the player feels central to the simulation, with the environment reacting realistically to their presence.

To ensure performance optimization, the implementation avoids unnecessary physics calculations by favoring CharacterControllers over Rigidbody-based movement. Colliders are kept minimal and efficient, and collision checks are streamlined for scenarios involving dense pedestrian populations. This approach enhances scalability, allowing the system to handle large numbers of pedestrians while maintaining a realistic and immersive experience.

By combining Unity's physics components with tailored scripting, the collision avoidance system enables natural, fluid interactions between pedestrians and the player, elevating the realism and immersion of the VR evacuation simulation.

**Pedestrian Groups and Behavior**

* **Groups Based on Speed and Destinations**: Pedestrians are categorized into different groups (e.g., walking and running) based on their speeds. They walk at a speed of 15 km/hr and run at a speed of 25 km/hr.
* **Waypoint-Based Navigation**: Each group follows unique sets of waypoints (waypoint arrays) and heads toward specific marshal points (evacuation destinations). The waypoints are created using small spheres as game objects spread out in a linear fashion, embedded just beneath the sketchup model of roads and pavements, hidden from the eyes of the player. They walk or run till the last waypoint and start walking or running backwards till the third last waypoint. This goes on in a loop and makes it look like a crowd gathered at the marshal point location. The 1st group are the ones going to marshal point 1, the 2nd goes to marshal point 2 and the 3rd goes to the other side towards the church.
* **Spawn point-based pedestrian generation**: All the pedestrian clones generated are from unique spawn points. There are a maximum of 60 spawn points, and all are located towards the front part of the building in front of the Halifax Central Library.
* **Random Spawn Intervals**: Pedestrians spawn at intervals randomized within a specific range (e.g., 1–1.1 seconds). This randomness ensures varied movement patterns and prevents overcrowding at spawn points. The interval is kept short to generate pedestrians faster as it would look like in an evacuation environment.

**Player and Pedestrian Interaction:** Pedestrians and the player avoid collisions through Unity’s physics system, primarily using **CharacterController, Rigid Body** and **Colliders**.When a collision is detected, the pedestrians adjust their direction, providing a natural avoidance behavior.

**Speaking Pedestrian Logic:** Speaking pedestrians use pre-recorded audio files. The audio is played dynamically, guiding the player toward the evacuation marshal point.The audio source is attached to the VR player headset/camera (XR Origin-XR Rig) and is played at intervals of 10 seconds to ensure unpredictability, creating a more engaging and realistic evacuation experience.The logic is implemented using Unity's AudioSource component, triggered via the respective script.

|  |
| --- |
| **Algorithm 1 – Pedestrian Movement** |
| Initialize variables (waypoints, speed), etc.  In Start():   * Initialize avoiding status and set the speed to meters per second * Get CharacterController reference   In Update():  If pedestrian is close to player:   * Call AvoidPlayer()   Else if another pedestrian is detected:   * Call AvoidPedestrian()   Else:   * If avoiding player or pedestrian, move toward waypoint * Else, move forward to the next waypoint   In AvoidPlayer():  If not avoiding player:   * Set avoidingPlayer to true * Set avoidTimer to avoidDuration * Randomly choose left or right for avoidance direction   Call PerformAvoidance()  In DetectOtherPedestrian():   * Detect nearby pedestrians * If detected, return true   In AvoidPedestrian():  If not avoiding pedestrian:   * Set avoidingPedestrian to true * Set avoidTimer to avoidDuration * Randomly choose left or right for avoidance direction   Call PerformAvoidance()  In PerformAvoidance():   * Keep Y position constant * Rotate towards avoidance direction * Move in the avoidance direction * If avoidTimer is up, stop avoidance   In MoveForward():  If there are more than one waypoint:   * Move towards the next waypoint * Rotate towards it   If close to the current waypoint, increment the waypoint index  Loop back to a specific waypoint if all are visited  In MoveTowardsWaypoint():  Similar to MoveForward(), but handles waypoint navigation and looping. |

Figure 6. – Pedestrian movement pseudo code

|  |
| --- |
| **Algorithm 2– Pedestrian Spawner** |
| Initialize Variables   * maleSpawnPoints, femaleSpawnPoints (spawn points for male and female pedestrians) * maleWaypointsGroup1, maleWaypointsGroup2, maleWaypointsGroup3 (waypoints for male pedestrians) * femaleWaypointsGroup1, femaleWaypointsGroup2, femaleWaypointsGroup3 (waypoints for female pedestrians) * minSpawnInterval, maxSpawnInterval (random spawn interval range) * maxPedestrians (maximum number of pedestrians allowed in the scene) * currentPedestrianCount (tracks number of pedestrians in the scene)   In Start():   * Initialize the spawn system. * Set the currentPedestrianCount to 0. * Start a spawning coroutine to spawn pedestrians at random intervals.   In Update():   * Continuously check if spawning conditions are met. * If the current number of pedestrians is less than maxPedestrians, allow spawning.   In SpawnPedestrian():   * If the current pedestrian count is less than the maximum: * Randomly select a spawn point (male or female spawn). * Instantiate a pedestrian at the selected spawn point. * Assign random waypoints and speed based on pedestrian type. * Randomly assign the pedestrian's group (Group1, Group2, or Group3). * Increment the pedestrian count.   In UpdatePedestrianCount():   * Decrement the currentPedestrianCount when a pedestrian is destroyed or reaches their destination. |

Figure 7. – Pedestrian spawning pseudo code

### 1.3.2 Fire

A static object on the Halifax Central Library. An audio file is integrated with the fire asset to produce realistic sound effects, adding to the immersive experience. Shader effects are applied for glow and flickering. Particle collisions ensure that fire particles interact with nearby objects for realism.

A blurry image of a fire

Description automatically generated

Figure 5. – Unity's fire asset

1.4 Player dynamics

The player’s movement integrates VR-specific inputs, such as thumbsticks for directional movement and button-based controls, ensuring a smooth locomotion experience. The system uses colliders to maintain physical realism, such as adhering to gravity and preventing the player from traversing outside the environment's boundaries. Pedestrian behavior dynamically guides the player to marshal points by adapting their paths and leading the player during evacuation scenarios.

Additionally, the game tracks player choices at key decision points (e.g., choosing between two evacuation routes) and logs these choices for analysis. Movement includes simulated head bobbing for added realism, smooth acceleration/deceleration, and distance tracking.

**Dynamic Player Guidance:** In scenarios without evacuation signs, the player follows the other pedestrians’ paths toward the marshal points.

**Player Choices and Feedback:** At the key junction, players are presented with two options for evacuation routes.The system tracks which path the player chooses, storing this information in the analytics pipeline for future analysis.

|  |
| --- |
| **Algorithm 3– VRMovement** |
| **Initialize Variables**   1. **Movement Settings:**  * speedKmH (player movement speed in km/h). * accelerationTime, decelerationTime (smoothing movement).  1. **Player Controller:**  * controller (CharacterController for the player). * xrRigTransform (Transform of the XR Rig).  1. **Tracking Variables:**  * currentVelocity (current player velocity for smooth transitions). * totalDistance (tracks total distance traveled by the player). * previousPosition (tracks the previous position for calculating distance moved).  1. **Destination Points:**  * destinationA and destinationB (marshal points). * playerChoice (indicates chosen destination: 1 for A, 2 for B). * distanceToA, distanceToB (store distances to each destination).  1. **Head Bobbing:**  * headBobbingEnabled, bobbingFrequency, bobbingAmplitude (simulate walking movement). * originalRigPosition (initial XR Rig position for head bobbing).   **In Start()**   1. Convert speedKmH to meters per second. 2. Initialize:  * currentVelocity to Vector3.zero. * previousPosition to the initial player position.  1. Save the XR Rig’s initial position for head bobbing. 2. Initialize distanceToA, distanceToB as large values.   **In Update()**   1. **Player Input:**  * Retrieve thumbstick and button input from the VR controllers. * Determine the movement direction based on input.  1. **Apply Movement:**  * Smooth acceleration or deceleration using Mathf.SmoothDamp. * Use the CharacterController.Move() method to update the player’s position.  1. **Head Bobbing:**  * Apply sinusoidal movement to simulate head bobbing if enabled and the player is moving.  1. **Clamp Height:**  * Restrict Y-position to prevent the player from falling below the ground.  1. **Calculate Distances:**  * Update distanceToA and distanceToB using Vector3.Distance() from the player to destinationA and destinationB. * Check if the player is within a threshold distance from either destination to determine playerChoice.  1. **Track Total Distance:**  * Add the distance moved since the last frame to totalDistance.   **In Move()**   1. Compute movement direction based on thumbstick input. 2. Smoothly transition the movement speed using acceleration or deceleration curves. 3. Update the player’s position using CharacterController.Move().   **In CheckDestinationChoice()**   1. Calculate distances from the player to both destinations. 2. Compare against a proximity threshold to determine which destination the player chooses. 3. Set playerChoice accordingly (1 for A, 2 for B).   **In ApplyHeadBobbing()**  If enabled and the player is moving:   1. Update the XR Rig’s Y-position using a sine wave function. 2. Ensure smooth transitions to simulate walking motion. |

Figure 8. – Player movement pseudo code

|  |
| --- |
| **Algorithm 4– PlayAudioAtIntervals** |
| **Initialize Variables**   * audioSource (Reference to the AudioSource component) * interval (Interval in seconds between audio plays) * timer (Timer to track when to play the audio)   **In Start():**   * If audioSource is not assigned, get the AudioSource component from the GameObject. * Set timer to the specified interval (or 0 for immediate playback).   **In Update():**   * Decrease the timer by Time.deltaTime each frame. * If the timer reaches 0 or less, play the audio and reset the timer to the interval value.   **In PlayAudio():**  If audioSource is assigned, play the audio |

Figure 9. – Playing audio file pseudo code

**Wheelchaired Player Dynamics:**

For the wheelchair-bound player, we first disable the XR Origin (XR Rig) which is the normal player. The wheelchaired movement dynamics are tailored to provide a realistic yet controlled experience. The key aspects include:

* **Button-Based Movement:** Unlike the standard player, the wheelchaired player uses buttons only (e.g., B and A for forward, X and Y for backward) to navigate the environment. It is crucial to communicate to the wheelchaired player to use only buttons for movement, ensuring a simulated wheelchaired movement.
* **Restricted Y-Axis Movement**: Movement is strictly on the horizontal plane, preventing unintended elevation changes.
* **Speed and Transition:** Movement speed is adjusted for the wheelchair's dynamics, with smooth acceleration and deceleration to ensure a natural flow.
* **Clamping Ground Position**: Like the standard player, the Y-position is clamped to prevent falling below the ground surface.

|  |
| --- |
| **Algorithm 5– Wheelchair Movement for Disabled Player** |
| **Initialization:**   * Define variables for speed (in km/h and m/s), acceleration, deceleration, and current velocity. * Initialize references for CharacterController and XR Rig.   **In Start():**   * Convert the movement speed from km/h to m/s.   **In Update():**   1. **Restrict Thumbstick Input:**    * Override thumbstick input for both left and right controllers, setting values to zero. 2. **Get Button Input for Movement:**    * Detect button presses (A, B, X, Y) for forward and backward movement.    * Compute the direction vector (buttonMovement). 3. **Smooth Acceleration and Deceleration:**    * Use linear interpolation to adjust the current velocity (currentVelocity) based on button input and acceleration/deceleration times. 4. **Move Wheelchair:**    * Convert the velocity into a movement vector and move the CharacterController.    * Ensure the movement is restricted to the horizontal plane (Y = 0). 5. **Prevent Falling Underground:**    * Clamp the Y position of the wheelchair to ensure it stays above the ground.   **RestrictThumbstickInput():**   * Check thumbstick input (primary2DAxis) for both left and right controllers. * Explicitly set the thumbstick values to zero to prevent movement.   **GetForwardButtonInput():**   1. **Initialize Movement Vector:**    * Start with a zero vector for movement (buttonMovement). 2. **Check Right Controller Buttons:**    * If B is pressed, add forward movement.    * If A is pressed, add backward movement. 3. **Check Left Controller Buttons:**    * If Y is pressed, add forward movement.    * If X is pressed, add backward movement. 4. **Return the Result:**    * Return the computed movement vector. |

Figure 10 - Pseudocode for Wheelchair Movement Script

# Evacuation scenarios development

The scenarios have been developed by employing the experimental design theory. The first step is to define the variables to be investigated. **Table 1** contains the employed variables and their levels in the experiment.

Table 1 – Levels of the variables

|  |  |  |  |
| --- | --- | --- | --- |
| Variable type | Variable name | Number of levels | Values |
| Context | Level of the fire | 2 | Half of the building /  Entire building |
| Evacuation signs | 2 | Not presented / Presented |
| Pedestrians speak mode | 2 | Do not speak / Speak |
| Alternative specific | Pedestrian flow | 2 | 30 ped / 50 ped |
| Pedestrian movement | 2 | Walking / Running |
| Trees presence at the evacuation marshal point | 2 | No / Yes |
| Bus presence at the evacuation marshal point | 2 | No / Yes |

Within the next step the full experimental plan has been generated. The plan considers variation of alternative specific variables for two marshal points which affects the full experimental plan size. Hence, comprising three context variables and eight alternative specific variables, the full experimental plan has a size of 2,048 experiments. Considering the sample-wise aspects, the full experimental plan should be reduced. The fractional design is utilized in this case to reach the necessary size of the experimental plan. A Fedorov exchange algorithm for D-optimal design has been employed that allows us to reduce the experimental plan up to 12 experiments (scenarios). The fractional experiment design is summarized in Table 2.

Table 2. – Scenarios for the outdoor evacuation virtual reality

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sce-nario | Context attributes | | | Marshall point’s #1 attributes | | | | Marshall point’s #2 attributes | | | |
| Building fire level | Evacua-tion signs | Spea-king mode | Pedest-rian flow | Pedest-rian movement | Trees | Bus | Pedest-rian flow | Pedestrian movement | Trees | Bus |
| 1 | Entire | No | Yes | 30 | Walking | Trees | Bus | 30 | Walking | No trees | No bus |
| 2 | Entire | Yes | No | 50 | Walking | No trees | No bus | 50 | Walking | No trees | No bus |
| 3 | Half | Yes | Yes | 50 | Running | Trees | No bus | 30 | Running | No trees | No bus |
| 4 | Half | No | Yes | 50 | Running | No trees | Bus | 50 | Walking | Trees | No bus |
| 5 | Entire | Yes | No | 30 | Running | No trees | Bus | 30 | Running | Trees | No bus |
| 6 | Half | No | No | 30 | Walking | Trees | No bus | 50 | Running | Trees | No bus |
| 7 | Half | No | No | 30 | Running | No trees | No bus | 30 | Walking | No trees | Bus |
| 8 | Half | Yes | Yes | 30 | Walking | No trees | Bus | 50 | Running | No trees | Bus |
| 9 | Entire | No | No | 50 | Running | Trees | Bus | 50 | Running | No trees | Bus |
| 10 | Half | Yes | No | 50 | Walking | Trees | Bus | 30 | Walking | Trees | Bus |
| 11 | Entire | Yes | Yes | 30 | Running | Trees | No bus | 50 | Walking | Trees | Bus |
| 12 | Entire | No | Yes | 50 | Walking | No trees | No bus | 30 | Running | Trees | Bus |

# Virtual reality evacuation experiment

Every participant of the evacuation experiment should evacuate in two separate scenarios. They are distinguished in the following way. The first scenario does not contain any evacuation signs, and the player should find the evacuation marshal point by interacting with game environment objects. Within this case, the speaking mode can be present that guides the player about the marshal point location. **Figure 10** depicts an example of the evacuation game within the first type of the scenario.

|  |  |
| --- | --- |
| a | b |
| c | d |

Figure 10. – An example of the evacuation sign-free scenario

**Figure 10a** – represents the start of the game when the player faces the dangerous situation, e.g., the building on the fire. This is an evacuation sign-free scenario. Hence, the player is guided by the movements of other pedestrians and their characteristics (see description of the scenarios planning in Table 1). **Figure 10b** depicts the pedestrian flows example. In turn, **Figure 10c** represents the junction where the player can choose between the two alternative marshal points to evacuate and **Figure 10d** shows the ultimate exit point for the player that they have reached the bus which will help them to evacuate.

|  |  |
| --- | --- |
| A building on fire in a city  Description automatically generated  a | b |
| c | d |

Figure 11. - An example of the evacuation sign-based scenario

**Figure 11a** – represents the beginning of the game when the player faces the dangerous situation, e.g., the building on the fire and sees the evacuation sign for them to vacate the area. **Figure 11b** depicts the pedestrian flows example. In turn, **Figure 11c** represents the bus stop location (one of the marshal points) with trees and is yet to choose which way to go in order to evacuate and **Figure 11d** shows the final exit location with a bus standing in front of a bus stop (the second alternative marshal point).

# Collected analytics

Every participant of the VR evacuation survey will provide two types of the datasets. The first one will be collected via the questionnaire procedure to be done before and after completion of the VR games. The second one will be collected directly during the VR evacuation scenarios. Havin two sources of the information coming from the same participant, every player will be provided with unique identification number allow us to match the datasets once the survey is finished. **Figure 12** represents the analytics collection process.

The VR evacuation scenarios analytics contains the following metrics collected:

1. Duration of the evacuation process for every played scenario.
2. Covered distance by the player within every evacuation scenario.
3. Chosen marshal point within every scenario.
4. Distances to both destinations (marshal points): The distance to each marshal point is calculated in real-time using Unity’s Vector3.Distance. The smaller of the two distances determines the nearest marshal point.

These data points are continuously tracked using custom Unity scripts.  
For instance:

float distanceToA = Vector3.Distance(controller.transform.position, destinationA.position);

float distanceToB = Vector3.Distance(controller.transform.position, destinationB.position);

Data is displayed to the player using Unity's UI console when the player quits the game. It is also saved in a CSV file using C# methods. The exported file includes rows for each player, scenarios and metrics.

Along with that, the pre- and post-game questionnaire will collect the participant-related information such as:

Pre-game questions:

* Basic socio-demographic information.
* Level of experience with VR.
* Experience with the evacuation situations.
* Self-assessment on stress resistance.
* Emotional state.
* Physical state.
* Environmental Comfort perceived in the lab.

Post-game questions:

* Perceived physical state after VR.
* Perceived level of danger during the VR evacuation.
* Easiness to utilize the VR tool.
* Perceived realistic of the fire and sounds.
* Scenario-wise evaluation aspects.
* Emotional state evaluation using SAM (self-assessment manikin).

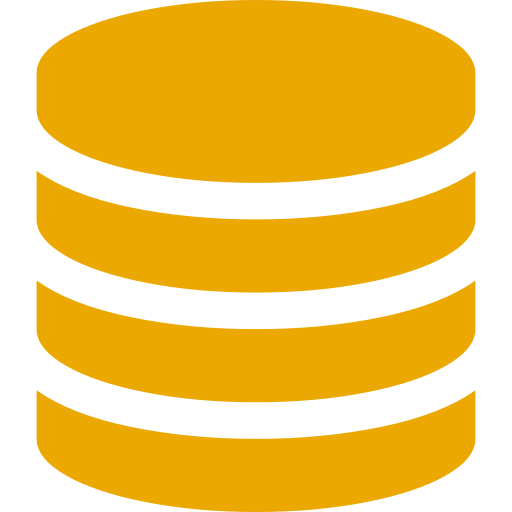
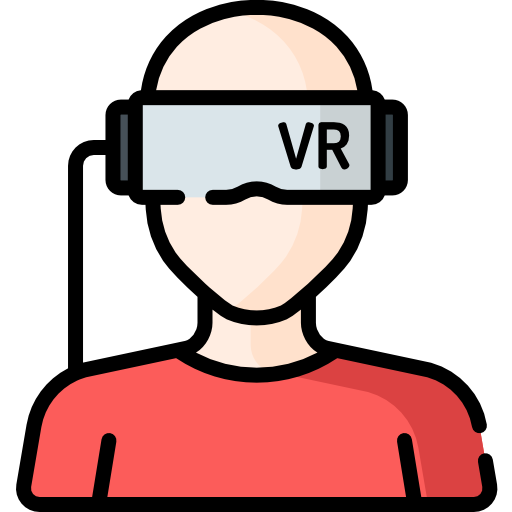
Annex 1 contains detailed information on the developed questionnaire for the pre- and post-game assessments.

Player *ID*

**Survey participants**

Pre- and post-survey questionnaire

VR Evacuation scenarios analytics



**Matching procedure**



**FINAL DATASET**

Figure 12. – Evacuation scenarios dataset collection procedure

# ANNEX 1 – Questionnaire

**----------------------------------------------------------------------------------------------------------------------------**

**INTRODUCTION to the VIRTUAL REALITY EVACUATION SIMULATIONS**

Dear participant, we highly appreciate their willingness to participate in our survey. It is aimed at exploring human behaviors under outdoor evacuation situations. The results will be used for the estimation the behavioral models acting as a tool for enhancing the evaluation procedures. The survey is completely anonymous and no personal information is collected.

During the survey they will be asked some **questions on the laptop** and also use the virtual reality headset to play in **two evacuation scenarios**. After completion the virtual reality experiment, they will be asked some questions regarding their **experience with virtual reality evacuation** as well as their **emotional conditions** during the simulations.

The virtual reality experiment can be stopped at any time upon their request.

**----------------------------------------------------------------------------------------------------------------------------**

**DESCRIPTION OF THE VIRTUAL REALITY EVACUATION TASK**

The player is being put into the outdoor fire evacuation situation. The player should evacuate from the initial spot and reach the **bus stop** which is considered as the evacuation marshal point. There are only **two evacuation marshal points** in the game which are located in specific locations, but not as on the real street.

The **game will be stopped** once you **reach** one of those two **marshal points**.

They will see **other pedestrians** in the game. These pedestrians **may be aware** of the marshal points **locations**.

The player is provided with **two joysticks** to mimic the walking/running process. Using only **right joystick** the **walking mode** is activated. Using **right and left joysticks simultaneously** will allow the player to speed up, i.e., to **run**.

**----------------------------------------------------------------------------------------------------------------------------**

**PRIOR the GAME questionnaire**

**GENERAL QUESTIONS:**

1. **What is their age?**
   1. 18-24
   2. 25-34
   3. 35-44
   4. 45-54
   5. 55-64
   6. 65 and more
2. **What is their gender:**
   1. Female
   2. Male
   3. Non-Binary
   4. Prefer not to disclose.
3. **What is their highest level of education achieved to date**:
   1. No certificate, no diploma or degree.
   2. High school certificate.
   3. College, CEGEP or other non-university certificate or diploma.
   4. University certificate or diploma below bachelor's level.
   5. University degree – Bachelor's.
   6. University degree – Master’s or PhD.
   7. Prefer not to disclose.
4. **What is their experience with the virtual reality:**
   1. Novice
   2. Beginner
   3. Intermediate
   4. Advanced
   5. Expert
5. **Did they have a life experience with an emergency evacuation** (not training experience or false alarms), such as in-door/outdoor fire evacuation, flood, hurricane, etc.:
   1. Yes
   2. No
6. **Do they have any fire-related phobias?**
   1. Yes
   2. No
7. **How can they evaluate themselves regarding the following statement**: “I am a stress resistant person and not panic during the extreme situations”?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |

**Please, evaluate the following statements about their current state:**

1. **Emotional state**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very nervous | Nervous | Neutral | Calm | Very calm |

1. **Physical state**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very poor | Poor | Average | Good | Excellent |

1. **Environmental Comfort perceived in the lab**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Uncomfortable | Uncomfortable | Neutral | Comfortable | Very Comfortable |

----------------------------------------------------------------------------------------------------------------------------

**POST-GAME questionnaire**

**GAME ENVIRONMENT-RELATED QUESTIONS:**

1. **I perceive negative consequences (side effects) from the app usage**:
   1. Yes
   2. No
2. **How close are they familiar with the studied street?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Not Familiar at all | Slightly Familiar | Moderately Familiar | Very Familiar | Extremely Familiar |

1. **Evaluate from 1 (no danger at all) to 10 (extremely dangerous) level of the danger perceived during the evacuation in the played game**:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No danger at all |  | | | | | | | | Extremely dangerous |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

1. **The app is easy to use**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |

1. **The fire flames represent the realistic conditions of the fire-related danger**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |

1. **The sound of the fire complements the perception of the fire-related danger**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |

1. **Did they find the trees as an obstacle to find the marshal point?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |

1. **Did they try to speed up during the evacuation procedure?**
   1. Constantly used the running mode.
   2. From time to time.
   3. Only in the beginning of the evacuation.
   4. Only at the end of the evacuation.
   5. Never.
2. **Please, evaluate the following aspects of evacuation scenarios that helped them to find and choose the marshal point:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Question** | This option **was not present** in my game | Not helped at all | Slightly helped | Moderately helped | Helped | Helped a lot |
| **Voice guidance** made by other pedestrians |  |  |  |  |  |  |
| **Signs** located on the pavement |  |  |  |  |  |  |
| **Number of people walking** to the specific evacuation marshal point |  |  |  |  |  |  |
| **Bus located** at the marshal point |  |  |  |  |  |  |
| **Crowd of people** located at the marshal point |  |  |  |  |  |  |

----------------------------------------------------------------------------------------------------------------------------

**EMOTION-RELATED QUESTIONS:**

**During the evacuation process how did they feel?**

1. **In terms of the situation control and uncertainty**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A black and white square with a figure  Description automatically generated** | | | | |  |
| **Completely uncertain** |  |  |  |  |  | **Completely confident in my action** |

1. **Perceived level of arousal/excitement when they reached the marshal point?**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A black and white logo  Description automatically generated** | | | | |  |
| **Relaxed/**  **Sleepy** |  |  |  |  |  | **Excited/ Aroused** |

1. **Perceived level of satisfaction/happiness with finding the marshal point**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
| **Unsatisfied/**  **Nervous** |  |  |  |  |  | **Satisfied/**  **Happy** |