VRScenarioBuilder: Free-Hand Immersive Authoring Tool for Scenario-based Testing of Automated Vehicles

Sevinc Eroglu[*]
Visual Computing Institute
RWTH Aachen University

Arthur Voigt
Visual Computing Institute
RWTH Aachen University

Benjamin Weyers [†]
Human-Computer Interaction
University of Trier

Torsten W. Kuhlen[‡]
Visual Computing Institute
RWTH Aachen University

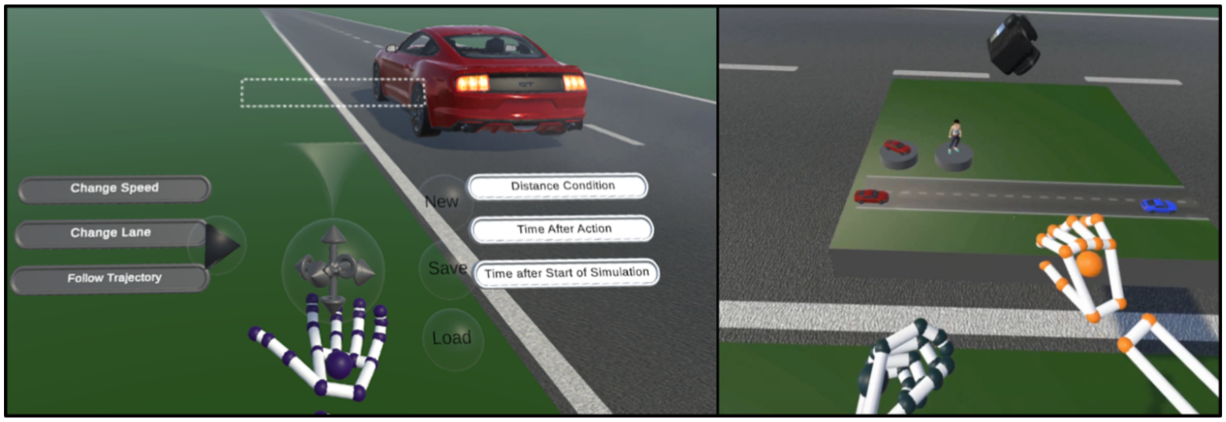

Figure 1: VRScenarioBuilder tool to author scenarios for testing automated driving features. **Left:** *Block Editor* for block-based assembly of dynamic traffic scenarios. **Right:** World-in-Miniature interface to navigate the scene, add and select entities (e.g. vehicle or pedestrian), and define the observer position during scenario playback.

Abstract

Virtual Reality has become an important medium in the automotive industry, providing engineers with a simulated platform to actively engage with and evaluate realistic driving scenarios for testing and validating automated vehicles. However, engineers are often restricted to using 2D desktop-based tools for designing driving scenarios, which can result in inefficiencies in the development and testing cycles. To this end, we present VRScenarioBuilder, an immersive authoring tool that enables engineers to create and modify dynamic driving scenarios directly in VR using free-hand interactions. Our tool features a natural user interface that enables users to create scenarios by using drag-and-drop building blocks. To evaluate the interface components and interactions, we conducted a user study with VR experts. Our findings highlight the effectiveness and potential improvements of our tool. We have further identified future research directions, such as exploring the spatial arrangement of the interface components and managing lengthy blocks.

Index Terms: Human-centered computing—Human computer interaction (HCI)—Virtual Reality; Human-centered computing—Interaction design and evaluation methods—User interface design—User studies

1 Introduction

Virtual Reality (VR) has gained increasing interest from the automotive industry, primarily due to its ability to provide simulated environments where engineers can actively participate and observe realistic driving scenarios. This enables engineers to test and validate automated vehicles in varying traffic scenarios without endangering themselves, vehicles, or the surrounding environment. By simulating environments and replicating real-world driving conditions, VR provides a controlled space for quick adjustments and improvements to vehicle systems. This accelerates development and testing cycles, leading to more efficient processes and reduced costs in the development of automated vehicle technologies.

Engineers often are limited to 2D desktop-based expert tools to design road networks and traffic scenarios for evaluation VR. However, even with expertise, it's inefficient to do a full design iteration and make intermediate changes, as it requires leaving the VR environment to apply modifications to the underlying system. This breaks the immersion and slows down the development and testing cycle.

A possible solution is to provide immersive authoring tools that enable users to create and modify virtual environments directly inside VR. This way, the user can design and evaluate the content without having to leave the virtual environment. To test automated vehicles, it is necessary to design road networks and define dynamic traffic scenarios. While previous works [16, 20] enable users to create and modify roads in an immersive environment, they do not provide the ability to author dynamic traffic scenarios.

To address these shortcomings, we present VRScenarioBuilder, an immersive test environment that enables engineers to efficiently test and validate automated vehicles in VR. To make interactions with the system as intuitive and approachable as possible, we have designed a natural user interface. Our tool enables users to create and modify dynamic traffic scenarios using free-hand gestures. To this end, we designed an easy-to-use drag-and-drop interface that is inspired by block-based visual programming languages. To evaluate our interface design decisions, we performed a user study with $n = 9$ VR experts. We gathered qualitative feedback to investigate potential difficulties and improve the proposed approach in future iterations. Furthermore, we identify future research directions based on the feedback and our observations.

[*]e-mail: eroglu@vr.rwth-aachen.de
[†]e-mail: weyers@uni-trier.de
[‡]e-mail: kuhlen@vr.rwth-aachen.de

2 Related Work

2.1 Virtual Testing of Automated Vehicles

To create testing scenarios for automated driving systems, road networks and traffic scenarios need to be defined. Real public data sources [12, 15, 26], procedural generation [28, 34], learning-based [13, 21, 33] or user-driven [6, 18, 38] methods are used to address these needs. User-driven methods offer a customizable approach that can be adjusted to meet specific needs, allowing for flexibility in the design process. Once a desired driving scenario has been identified, the user manually specifies parameters, including coordinates, trajectories, velocities, and orientations using a domain-specific language (DSL) such as ASAM OpenSCENARIO [5]. This is an open XML-based standard describing dynamic content for virtual test drive simulations, containing the maneuvers of traffic participants. These scenarios, utilized as input for simulators in testing automated driving functions, support reusability across tools and development stages. Thus, engineers commonly use simulators like CARLA [17], IPG CarMaker [8], Cognata [9], SVL [10], and DYNA4 [1]. However, these simulators do not enable users to modify the traffic scenarios interactively while being immersed. Eroglu et al. [20] have addressed this aspect by providing an immersive authoring tool that enables engineers to test automated vehicles. In their work, users can create road networks and traffic scenarios using free-hand interactions. While they offer features for designing traffic scenarios, such as adding traffic lights, defining traffic directions, and adding speed limits, they do not provide the rule-based traffic scenario definition that we propose in our work.

2.2 Dynamic Content Description in VR

Dynamic content that reacts to other scene objects based on a defined rule is typically described using programming languages. To create the content while being immersed, researchers have adapted visual programming languages (VPL) rather than textual programming to avoid the inconvenience of using a physical keyboard in VR [19, 25]. VPLs use visual elements such as graphical symbols and diagrams to represent and manipulate code structures. They enable users to create programs without the need for extensive programming skills or knowledge. Among these, one of the immersive VPL expands upon block-based programming paradigm [24, 31, 36, 39], which enables users to compose a program by positioning blocks that represent basic commands and control flow structures. Both Cubely [36] and VR-ocks [31] are primarily used for educational purposes to introduce novice programmers to basic programming concepts. Jin et al. [24] proposed VWorld, which enables children to learn programming using blocks in VR. Similar to our approach, the authors utilize a WIM to manipulate the virtual environment. To teach students "Internet-of-Things" knowledge through block-based visual programming, Zhu et al. proposed LearnIoTVR [39]. Furthermore, Hedlund et al. [23] propose BlocklyVR, investigating the potential and constraints of using block-based programming in VR environments, where users engage in programming activities by physically moving, in contrast to the conventional desktop-based 2D Blockly [7] situated context. Both BlocklyVR and LearnIoTVR use the interface design of Blockly, which allows for horizontal and vertical nesting of blocks.

Our interface design, presented for creating dynamic traffic scenarios, inspired a block-based VLP approach similar to the present works. However, we further simplified the programming process by using pre-defined blocks of conditions and actions rather than basic building blocks. While our design enables users to stack the blocks vertically, the additional parameter adjustments within these blocks are made through button presses, sliders, and demonstration via direct interaction, such as defining a trajectory for the entity to follow.

3 VRScenarioBuilder

We developed our system based on the following requirements that are defined by control engineering experts of an automotive company.

- **R1** The system should enable users to create, modify, and fine-tune dynamic traffic scenarios that include driver actions such as changing a lane, overtaking or following a defined trajectory.
- **R2** The system should enable users to observe and analyze scenarios from different perspectives for comprehensive testing.
- **R3** The system should facilitate offline editing and compatibility with widely used standards like OpenSCENARIO.

To address each requirement, we designed an immersive authoring system with two main user interface components: the *World-in-Miniature* (Section 3.1) and the *Block Editor* (Section 3.2). In the following, we present our system in more detail.

3.1 World-in-Miniature

To enable users to efficiently interact with virtual objects within arm's reach, we utilize a World-in-Miniature (WIM) interface [32], a miniaturized replica of the virtual environment. Upon opening the right hand with the palm facing upward, the WIM interface becomes visible in front of the user. To enable users to interact effectively with the virtual environment from various perspectives, the WIM can be freely repositioned with a single hand and rotated around the vertical axis with both hands. To change the scene inside of the WIM, we enable users to employ a swiping gesture on the floor, similar to scrolling on a smartphone.

The first step in designing a traffic scenario (**R1**) is to add entities to the virtual environment. Currently, our system provides two different entities: *Vehicle* and *Pedestrian*. The miniature copies of these entities are located in their respective container on the WIM interface (See Figure 1 - **Right**). These objects can be added to the scene by grabbing and placing them in the WIM. After removing a copy from its container, a new copy of that entity appears in its place.

The position and orientation of scene objects can be manipulated by moving them inside the WIM. To reduce the authoring time, the position and the orientation of vehicles are automatically adjusted to fit the nearest lane on release. Pedestrians, on the other hand, can be positioned and oriented arbitrarily.

To address **R2**, the WIM interface incorporates a camera that enables users to define their initial position and viewing direction for testing the scenario. During authoring phase, the camera can be moved by grabbing it and rotated by manipulating the lens. Via the preview screen of the camera, users can determine their perspective within the scenario. The camera can also be used to follow a specific entity within the simulation. This can be accomplished by dragging the camera onto the entity of interest and releasing it. The camera will then "snap" to the entity and become smaller, indicating to the user that it is now following that entity during the simulation.

3.2 Block Editor

In order to design an easy-to-use interface for authoring dynamic traffic scenarios, we needed to simplify the complex hierarchy of elements in OpenSCENARIO, which can require 258 lines of code to define a simple lane-cutting scenario [27]. To achieve this simplicity, we drew inspiration from the block-based visual programming paradigm. In this approach, code snippets are represented by blocks that can be dragged together to create programs. This provides a simple way for program creation that doesn't demand extensive programming skills or knowledge. To this end, our *Block Editor* user interface component enables users to author dynamic traffic scenarios by dragging and dropping building blocks, namely condition and action blocks (**R1**).

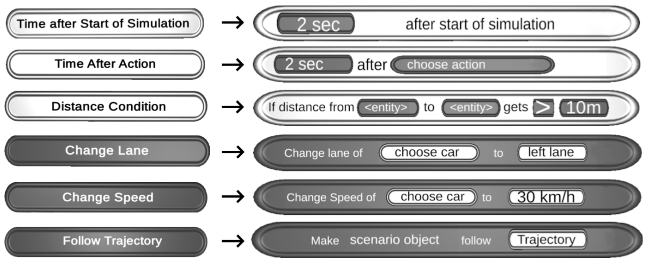

Figure 2: This figure shows how each block changes in appearance once it has been added to the programming area. This change in appearance causes the blocks to have different parameters to specify the condition/action. These parameters are highlighted black for the white *Condition Blocks* and white for the black *Action Blocks*.

The Block Editor can be brought into the virtual environment by turning the left hand with the palm facing upwards. To enable users to comfortably interact with the interface, it can further be repositioned freely by grabbing its handle, drawn as a sphere containing 3D arrows. Additionally, while positioning, the interface rotates in a way that directly faces the user, to ensure optimal visibility and avoid additional interactions that could potentially cause arm fatigue [22].

3.2.1 Assembling Blocks

The main UI elements of the Block Editor component are *Condition Blocks* and *Action Blocks*. Condition Blocks are similar to "if" statements in programming languages, and Action Blocks serving as function calls within these "if" statements. Once a condition is met, the corresponding actions are executed. Currently, Condition Blocks can only exist on the first level and Action Blocks on the second. This setup allows for multiple dependent action blocks per condition block. However, each condition block is evaluated independently of others on the first level during playback.

The Block Editor currently offers three types of Condition Blocks and three types of Action Blocks. The functionalities of these are designed to create dynamic scenarios such as "Cut-In" [2], "Double Lane Changer" [3], and "Overtaker" [4], as required by the system control engineers (**R1**). To define these scenarios, we have designed the following blocks:

Condition Blocks:

- *Distance* is triggered when the distance between two selected entities becomes greater or smaller than a defined threshold in meters.
- *Time After Action* is triggered when the defined time has passed after the execution of a specified action.
- *Time After Start* is triggered when the defined time has passed after the simulation started.

Action Blocks:

- *Change Speed* sets the desired speed of a selected vehicle to a specified one.
- *Change Lane* causes a selected vehicle to change its lane to a selected (right or left) lane.
- *Follow Trajectory* make a chosen entity, a vehicle or a pedestrian, to follow a defined trajectory.

To create a scenario, the blocks need to be positioned in input area, denoted by a white dashed rectangle in the upper-middle part of the interface (See Figure 1 - **Left**). Upon adding a block, all exiting blocks move upwards to maintain space in the input area for the next condition or action block. Simultaneously, its appearance expands to display adjustable parameters required for further specification (See Figure 2). To remove a block from the scenario, the block can simply be grabbed and thrown away. The block will fall and be deleted.

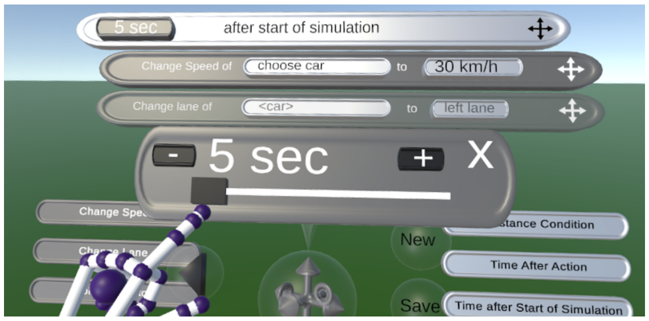

Figure 3: Depiction of VRScenarioBuilder's *Slider*. The user clicked on the "5 sec" button of the *Time After Start* condition block. The parameter can be set by moving the slider with the index finger. The two buttons with "-" and "+" can be used for fine adjustment.

3.2.2 Parameter Specification

To enable users to define parameters, our system offers 3D buttons and sliders. When a button on the expanded block is pressed, the system performs a corresponding action based on the parameter type. For numerical parameters, users can define the value by interacting with a slider. For a precise definition of the values, +/- buttons can be employed (See Figure 3). Additionally, we provide toggles for binary parameters that are used in the *Distance* condition ("$>$" or "$<$") and the *Change Lane* action blocks ("left lane" or "right lane").

The *Change Lane* and the *Chance Speed* blocks require the user to specify which vehicle should execute the action. To perform this selection of a vehicle, the WIM interface is employed. After pressing the "choose car" button, the Block Editor disappears and the WIM appears in front of the user, and the vehicle can be selected by touching it directly with the index finger. The *Distance Condition* and *Follow Trajectory* blocks have a similar parameter, "entity", allowing the selection of a pedestrian or a vehicle. The *Time After Action* block incorporates a parameter defining which action triggers this condition. To set this parameter, the user needs to press the button and then select the desired action.

Lastly, as a parameter of *Follow Trajectory*, a trajectory can be defined. For this reason, we enable users to draw a path by directly grabbing the object (a vehicle or a pedestrian) and moving it in the WIM (See Figure 4). The resulting path is projected onto the road and is visualized in white to provide visual feedback.

3.2.3 Playing Scenarios

After a scenario has been created by assembling condition and action blocks, simulation can be initiated by interacting with the play button on the Block Editor interface. Upon play, the position of the user will be set based on the miniature camera. The position of the user will be updated to follow an entity if the camera is attached to that entity. During the simulation mode, the Scenario Builder is adapted to offer new input capabilities to pause, resume, and stop the scenario. For clear observation of the simulation steps, annotations are provided to indicate ongoing actions (See Figure 5). Their transformation dynamically updates based on the user's position and they disappear after some seconds.

3.2.4 Dynamic Editing of Scenarios

Our system further enables users to dynamically modify scenarios while the simulation is running (**R1**). It's a crucial feature for eval-

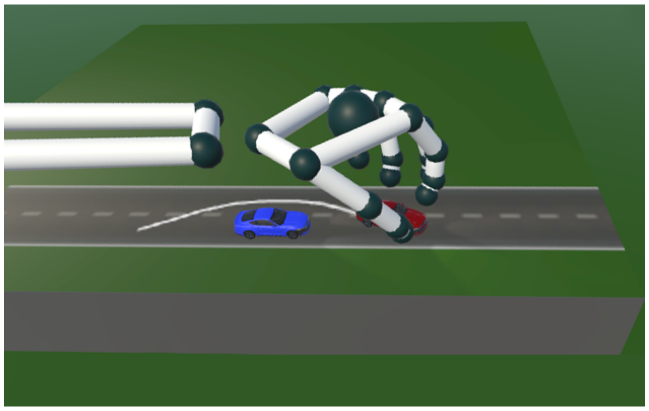

Figure 4: Creating a trajectory in the WIM: The user is moving the red car, creating a trajectory to overtake the blue car.

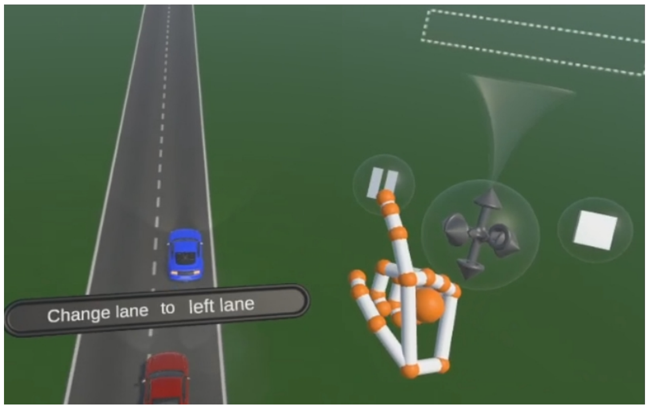

Figure 5: Playback of a scenario: The Block Editor displays two buttons to pause or stop the simulation. Over the simulated vehicle, an annotation indicates the red car is changing to the left lane.

uating how individual entity behaviors affect scenario outcomes at specific points in the simulation. To achieve this, the simulation first needs to be paused and then desired action blocks can be added to the input area of the Block Editor. Once the modifications have been made, the user can then resume the simulation and observe the effects of the changes.

3.2.5 Saving and Loading of Scenarios

Another crucial feature requested by engineers is the ability to modify the scenarios offline and share them for use in other testing environments (**R3**). For this reason, we enable users to save scenarios in the XML format of OpenSCENARIO. It is commonly employed in the automotive industry and compatible with various testing environments such as CarMaker [8] or CARLA [17]. A created scenario can be saved by selecting the save button on the Block Editor (see Figure 1 - **Left**). The scenario file will be stored under the project folder. It can be modified offline and, e.g., loaded into a desktop-based testing environment. The modified file can then be loaded into our tool to continue the workflow in VR.

4 User Study

To gather initial feedback on the interface components and free-hand interactions of VRScenarioBuilder, we conducted a user study with VR experts. This study serves as an initial guide for refining the interface concept, enhancing usability, and implementing additional features.

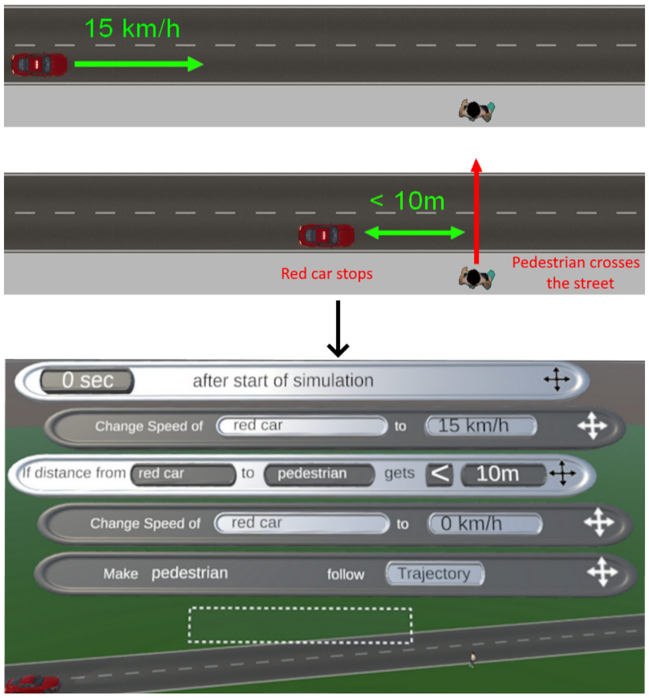

Figure 6: **Top:** The scenario depiction shown to users to recreate in the study: The red car begins to drive at 15km/h, if the distance to the pedestrian is less than 10m, the car stops and the pedestrian crosses the street. **Bottom:** The depiction of correctly assembled blocks and set parameters for the scenario described above.

4.1 Apparatus

The study was conducted in a seated position at our lab. We used the HTC Vive HMD with two Lighthouse 1.0 base stations. In order to track the position and orientation of the user's fingers and hands, we employed the Leap Motion Controller (LMC) [11], mounted onto the VR headset. The experimental platform was developed in Unity 2018.3.14f1 and ran on a 3.50GHz Intel Xeon E5-1650 with NVIDIA GeForce GTX 1080 GPU, operating Windows 10.

4.2 Procedure

The study procedure is composed of several sequential steps. First, participants signed a consent form and filled out a pre-study demographic questionnaire. Then, an example OpenSCENARIO file "Overtaker.xml" and the VRScenarioBuilder workflow were shown and briefly explained.

Once equipped with the HMD, participants were instructed to think aloud and freely comment on the system. After getting familiar with the system, the participants were instructed to load the example file using the Block Editor and play this scenario.

In the next step, the participants had to create a new scenario based on the image that is shown in VR (See Figure 6 - **Top**). Several features were utilized in this task, including block assembly, parameter configuration, as well as creating a trajectory for the pedestrian. Upon successful creation (See Figure 6 - **Bottom**), the participants were instructed to play this scenario.

Once the pedestrian had crossed the street, the participants were instructed to use the dynamic editing feature to make the car continue driving. This involved pausing the scenario, adding a *Change Speed* action, setting the "choose car" parameter, and resuming playback.

After completing the tasks, participants put the HMD off and completed the System Usability Scale (SUS) [14], a custom 5-point Likert scale questionnaire (See Figure 7). Furthermore, semi-structured interviews were conducted with each participant to gather feedback on system difficulties and suggestions for improvement. The entire procedure took approximately 45 min.

4.3 Participants

9 persons participated in the study. They were between the ages of 22 and 33 (7 male and 2 female). All participants had normal or corrected vision. None of them suffered from color blindness, and all were right-handed. Additionally, all participants reported having experience with VR and 3D user interfaces regularly, with 5 reporting previous experience with free-hand interaction in VR. All participants also reported having regular programming experience.

4.4 Results and Discussion

Overall, we mainly received positive feedback from VR experts. They were able to solve the tasks without major problems and rated the system's usability as good, giving it an average SUS score of 75. Furthermore, they found the interaction with the system using free-hand gestures was *not physically exhausting* **Q1** and *functioned well* **Q10**. The further results of the custom 5-point Likert scale questionnaire can be seen in Figure 7. The experts further provided valuable feedback on existing and suggested additional features, which we present and discuss in the following.

4.4.1 Feedback on Existing Features

Concerning loading of scenarios, several participants commented positively on the scenario representation using blocks as being *easy to understand*. Upon playing the loaded scenario, one participant stated that the annotations *could sometimes cover important elements* of the scenario. To address this, we plan to implement an automated annotation positioning approach to prevent interference with important elements of the scenario, similar to previous work [29, 35].

Regarding scenario creation, users expressed confidence in using the system and appreciated the block assembly concept for its *intuitiveness*. These positive responses are further supported by the ratings of **Q2** and **Q7** in the custom questionnaire (See Figure 7). Two participants, however, experienced issues when attempting to insert a block into the programming area. The block would sometimes *fall down accidentally* when released too early. To solve this issue, we plan to provide improved visual feedback to indicate when it is safe to release the block. In addition, we also plan to implement an undo/redo feature to enable users to correct accidental actions.

In terms of parameter specification, several participants gave positive feedback on the *ease of using* the UI elements of the Block Editor to set numerical parameters. However, one participant stated that the transition from the Block Editor and the WIM was sometimes confusing when selecting a "entity" parameter. To improve the user experience, we plan to evaluate different approaches for transitioning between the interfaces.

4.4.2 Ideas for Additional Features

We received valuable insights for improving the functionality of VRScenarioBuilder. One notable suggestion is to enable the *combination of multiple conditions* using a logical "and" by nesting consecutive Condition Blocks directly below one another. Another suggestion from two participants was to have a *list of scenario entities* that can be selected if the WIM becomes crowded. Furthermore, regarding the playback of scenarios, one participant suggested a feature to *"scroll in time"* during playback to enable users to revisit specific moments in a scenario. As part of future design iterations, we will carefully consider these suggestions.

5 Lessons Learned & Future Research Directions

Several key lessons have emerged from our observations and the feedback that we gathered.

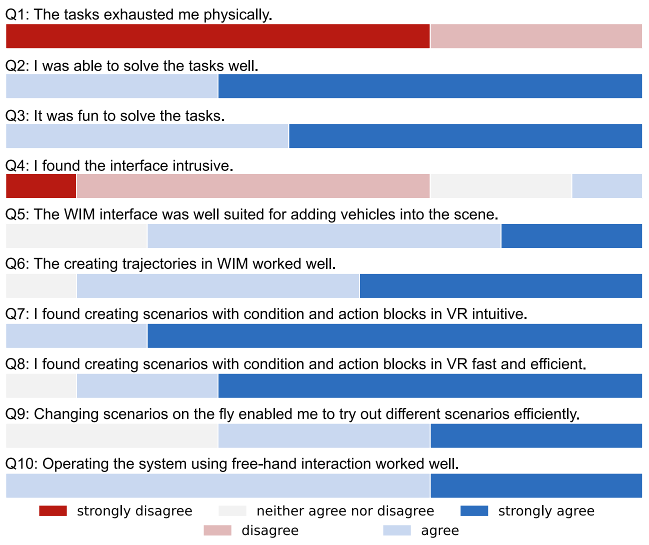

Figure 7: Histogram of answers to a custom 5-point Likert scale questionnaire.

- **Extension to Multimodal Cues:** We have observed that our system is lacking some important visual cues within the user interface such as indicating where to safely add blocks. In addition to visual cues, carefully selected auditory cues can also be used to further enhance the user experience [30]. In order to improve user understanding, reduce confusion, and simplify interaction with the system, we plan to incorporate auditory and more prominent visual cues in future iterations.

- **Investigating the Layout of the Interfaces:** When specifying entities as parameters, the WIM interface appears and the Block Editor disappears until an entity has been selected. This design decision was made to avoid potential clutter in the user's view. However, based on feedback we have received, the transition between the Block Editor and the WIM interface can be confusing. This can potentially lead to accidental selection of the wrong entity, especially when the WIM is crowded. Simultaneously displaying the Block Editor and the WIM interfaces may introduce challenges related to visual clutter and optimal positioning within the interaction range of the user. To tackle this issue, our research will focus on investigating the following research question:

  RQ1: How can the spatial positioning and the visibility of user interface elements be designed to minimize visual clutter and improve the user experience during scenario creation in VR?

  To address the question in future work, we plan to incorporate additional quantitative measures besides task completion time and cognitive load. For instance, applying the dual-task paradigm [37] would enable us to measure cognitive load during the use of the user interface. This will help us gain a deeper understanding of which elements of the user interface affect cognitive load in particular. In this way, it could be possible to examine in more depth how spatial design affects the user experience while creating immersive scenarios.

- **Lengthy Scenario Code Blocks:** We observed that as the number of blocks increases in the Block Editor interface, there could be a possible challenge in terms of navigation, readability, and efficient management. To handle this complexity, we propose the following solutions:

  Implementing a *hierarchical structure that organises blocks into categories* could simplify the navigation process. This way users can collapse or expand sections, allowing for a more focused and efficient exploration of scenario blocks. Additionally, *search* and *filter* functionalities can be added to the interface to enable users to quickly locate specific blocks. This feature may allow users to manage extensive lists more easily by providing targeted access based on naming or functional attributes. Offering users the ability to create *custom groupings* within the interface could be another solution. This would allow users to organise scenario blocks based on their preferences and workflow, which could result in an experience that is intuitive and easy to use. Building upon these proposed solutions, we have developed the following research question:

RQ2: How can an interface be designed to manage and navigate an extensive list of scenario code blocks efficiently in VR?

6 Conclusion

We presented VRScenarioBuilder as a novel solution to address the limitations imposed by 2D desktop-based tools in designing and testing dynamic driving scenarios for automated vehicles in VR. Our tool enables users to create dynamic traffic scenarios and modify them at run-time based on free-hand gestures. By performing design and evaluation steps within the application without a break of immersion, it offers an efficient workflow for testing automated driving features. We designed an intuitive drag-and-drop scenario builder interface, drawing inspiration from block-based visual programming languages for user-friendly interaction.

To evaluate our interface design choices, we conducted a user study with VR experts. Furthermore, we gathered qualitative feedback to find potential challenges and refine our approach in future iterations. The results show the intuitiveness of design decisions for creating dynamic scenarios with free-hand interactions, with the participants rating the system's usability as good, thereby validating the effectiveness of our approach. In addition, we identified future research directions based on feedback and our observations. These aim to enhance the interface and further contribute to the improvement of immersive scenario authoring for testing automated driving features in VR.

Acknowledgments

The authors express their gratitude to Alain Chevalier and Frederic Stefan for their insightful discussions that greatly contributed to defining the requirements of our tool.

References

[1] Adas testing with virtual test drives — vector. `https://www.vector.com/int/en/products/products-a-z/software/dyna4/adas-testing-with-virtual-test-drives/`. (Accessed on 01/14/2024).

[2] Asam openscenario: User guide. `https://releases.asam.net/OpenSCENARIO/1.0.0/ASAM_OpenSCENARIO_BS-1-2_User-Guide_V1-0-0.html#_cut_in`. (Accessed on 11/29/2023).

[3] Asam openscenario: User guide. `https://releases.asam.net/OpenSCENARIO/1.0.0/ASAM_OpenSCENARIO_BS-1-2_User-Guide_V1-0-0.html#_double_lane_changer`. (Accessed on 11/29/2023).

[4] Asam openscenario: User guide. `https://releases.asam.net/OpenSCENARIO/1.0.0/ASAM_OpenSCENARIO_BS-1-2_User-Guide_V1-0-0.html#_overtaker`. (Accessed on 11/29/2023).

[5] Asam openscenario® dsl. `https://www.asam.net/standards/detail/openscenario-dsl/`. (Accessed on 01/11/2024).

[6] asam-oss/osc-alks-scenarios: Alks scenario interpretation in openscenario. `https://github.com/asam-oss/OSC-ALKS-scenarios`. (Accessed on 01/12/2024).

[7] Blockly — google for developers. `https://developers.google.com/blockly`. (Accessed on 01/13/2024).

[8] Carmaker — ipg automotive. `https://ipg-automotive.com/en/products-solutions/software/carmaker/`. (Accessed on 01/11/2024).

[9] Cognata — autonomous and adas vehicles simulation software. `https://www.cognata.com/`. (Accessed on 01/11/2024).

[10] Svl simulator by lg - autonomous and robotics real-time sensor simulation, lidar, camera simulation for ros1, ros2, autoware, baidu apollo. perception, planning, localization, sil and hil simulation, open source and free. `https://www.svlsimulator.com/`. (Accessed on 01/11/2024).

[11] Ultraleap. https://www.ultraleap.com/leap-motion-controller-whats-included/. (Accessed on 11/30/2023).

[12] C. L. Azevedo, N. M. Deshmukh, B. Marimuthu, S. Oh, K. Marczuk, H. Soh, K. Basak, T. Toledo, L.-S. Peh, and M. E. Ben-Akiva. Simmobility short-term: An integrated microscopic mobility simulator. *Transportation Research Record*, 2622(1):13–23, 2017.

[13] L. Bergamini, Y. Ye, O. Scheel, L. Chen, C. Hu, L. Del Pero, B. Osiński, H. Grimmett, and P. Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5119–5125. IEEE, 2021.

[14] J. Brooke. SUS -A quick and dirty usability scale Usability and context. *Usability evaluation in industry*, 1996.

[15] P. Cai, Y. Lee, Y. Luo, and D. Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4023–4029. IEEE, 2020.

[16] S. Côté and O. Beaulieu. VR Road and Construction Site Safety Conceptual Modeling Based on Hand Gestures. *Frontiers in Robotics and AI*, 6:15, 2019.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pp. 1–16. PMLR, 2017.

[18] D. drive Contributors. DI-drive: OpenDILab decision intelligence platform for autonomous driving simulation. `https://github.com/opendilab/DI-drive`, 2021.

[19] J. Dudley, H. Benko, D. Wigdor, and P. O. Kristensson. Performance envelopes of virtual keyboard text input strategies in virtual reality. In *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 289–300. IEEE, 2019.

[20] S. Eroglu, F. Stefan, A. Chevalier, D. Roettger, D. Zielasko, T. W. Kuhlen, and B. Weyers. Design and evaluation of a free-hand vr-based authoring environment for automated vehicle testing. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 1–10. IEEE, 2021.

[21] L. Feng, Q. Li, Z. Peng, S. Tan, and B. Zhou. Trafficgen: Learning to generate diverse and realistic traffic scenarios. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3567–3575. IEEE, 2023.

[22] J. Hansberger, C. Peng, S. Mathis, V. Areyur Shanthakumar, S. Meacham, L. Cao, and V. Blakely. Dispelling the gorilla arm syndrome: The viability of prolonged gesture interactions. pp. 505–520, 07 2017. doi: 10.1007/978-3-319-57987-0_41

[23] M. Hedlund, A. Jonsson, C. Bogdan, G. Meixner, E. Ekblom Bak, and A. Matviienko. Blocklyvr: Exploring block-based programming in virtual reality. In *Proceedings of the 22nd International Conference on Mobile and Ubiquitous Multimedia*, pp. 257–269, 2023.

[24] Q. Jin, Y. Liu, Y. Yuan, L. Yarosh, and E. S. Rosenberg. Vworld: an immersive vr system for learning programming. In *Proceedings of the 2020 ACM Interaction Design and Children Conference: Extended Abstracts*, pp. 235–240, 2020.

[25] P. Knierim, V. Schwind, A. M. Feit, F. Nieuwenhuizen, and N. Henze. Physical keyboards in virtual reality: Analysis of typing performance and effects of avatar hands. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pp. 1–9, 2018.

[26] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pp. 2575–2582. IEEE, 2018.

[27] G. Lou, Y. Deng, X. Zheng, M. Zhang, and T. Zhang. Testing of autonomous driving systems: where are we and where should we go?

In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 31–43, 2022.

[28] I. Paranjape, A. Jawad, Y. Xu, A. Song, and J. Whitehead. A modular architecture for procedural generation of towns, intersections and scenarios for testing autonomous vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 162–168. IEEE, 2020.

[29] S. Pick, B. Hentschel, I. Tedjo-Palczynski, M. Wolter, and T. W. Kuhlen. Automated positioning of annotations in immersive virtual environments. In *EGVE/EuroVR/VEC*, pp. 1–8, 2010.

[30] R. Schlünsen, O. Ariza, and F. Steinicke. A vr study on freehand vs. widgets for 3d manipulation tasks. In *Proceedings of Mensch Und Computer 2019*, pp. 223–233. 2019.

[31] R. J. Segura, F. J. del Pino, C. J. Ogáyar, and A. J. Rueda. VR-OCKS: A virtual reality game for learning the basic concepts of programming. *Computer Applications in Engineering Education*, 2020. doi: 10.1002/cae.22172

[32] R. Stoakley, M. J. Conway, and R. Pausch. Virtual reality on a WIM: interactive worlds in miniature. In *Conference on Human Factors in Computing Systems - Proceedings*, 1995.

[33] Q. Sun, X. Huang, B. C. Williams, and H. Zhao. Intersim: Interactive traffic simulation via explicit relation modeling. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11416–11423. IEEE, 2022.

[34] S. Tan, K. Wong, S. Wang, S. Manivasagam, M. Ren, and R. Urtasun. Scenegen: Learning to generate realistic traffic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 892–901, 2021.

[35] M. Tatzgern, D. Kalkofen, R. Grasset, and D. Schmalstieg. Hedgehog labeling: View management techniques for external labels in 3d space. In *2014 IEEE Virtual Reality (VR)*, pp. 27–32. IEEE, 2014.

[36] J. Vincur, M. Konopka, J. Tvarozek, M. Hoang, and P. Navrat. Cubely: Virtual reality block-based programming environment. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST*, 2017. doi: 10.1145/3139131.3141785

[37] C. D. Wickens. Processing resources in attention, dual task performance, and workload assessment. 1981.

[38] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang. Specification-based autonomous driving system testing. *IEEE Transactions on Software Engineering*, 2023.

[39] Z. Zhu, Z. Liu, Y. Zhang, L. Zhu, J. Huang, A. M. Villanueva, X. Qian, K. Peppler, and K. Ramani. Learniotvr: An end-to-end virtual reality environment providing authentic learning experiences for internet of things. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–17, 2023.