VReqDV: Model Based Design Generation & Design Versioning Tool for Virtual Reality Product Development

Shambhavi Jahagirdar[a], Sai Anirudh Karre[b] and Y. Raghu Reddy[c]
SERC, IIIT Hyderabad, India

Keywords: Virtual Reality, Design Versioning, Design Generation, Meta-Model, Requirements Specification, Traceability.

Abstract: Virtual Reality (VR) product requires expertise from diverse set of stakeholders. Moving from requirements to design mock-up(s) while building a VR product is an iterative process and requires manual effort. Due to lack of tool support, creating design templates and managing the respective versions turns out to be laborious and difficult. In this paper, we describe VReqDV, a model-driven VR design generation and versioning tool that can address this gap. The tool uses VR meta-model template as a foundation to facilitate a design pipeline. VReqDV can potentially facilitate design generation, design viewing, design versioning, design to requirements conformity, traceability, and maintenance. It is a step forward in creating a Model-Driven Development pipeline for VR scene design generation. We demonstrate the capabilities of VReqDV using a simple game scene and share our insights for wider adoption by the VR community.

1 MOTIVATION

The development of Virtual Reality (VR) products is inherently complex and time consuming, requiring collaboration among stakeholders with expertise in diverse aspects. Minor design changes can cascade into significant development delays (Mattioli et al., 2015). Research has indicated that the design phase is critical in the VR product development lifecycle, requiring prompt sign-off to mitigate the risk of delivery delays (Karre et al., 2019). Current VR design pipelines employ manual, iterative methods and lack robust version control, thereby hindering collaborative efficiency.

Some of the key challenges in VR design include (1) managing complex scene layouts, (2) positioning of articles, (3) coordinating object interactions and behaviors, and (4) accommodating diverse user inputs. Each of these challenges is further enhanced by the unique attributes of VR technology, such as spatial dynamics and the necessity for heightened user immersion, which collectively necessitate responsive design strategies. The reliance on manual design iterations often results in inefficiencies and inconsistencies that complicate version control and traceability (Troyer et al., 2009). As projects transition from two-dimensional (2D) to three-dimensional (3D) environments, the complexity of user interactions escalates substantially, necessitating a comprehensive set of properties to accurately describe virtual objects. Consequently, incomplete or under-specified requirements may lead to numerous design variations that are susceptible to different interpretations (Geiger et al., 2000a). This phenomenon creates a cycle of conformance challenges between requirements and design. Such observations are particularly relevant to VR technology, as it possesses a unique product development cycle (Balzerkiewitz and Stechert, 2021) and the VR domain itself is characterized by multimodal use cases, distinct stakeholders roles, subpar tool support, and volatile hardware requirements. As a result, alignment between requirements and design is crucial to avert unexpected development costs.

Current industry practices of VR design remain largely manual and semi-automated (Cao et al., 2023)(Wu et al., 2021). Development teams typically rely on iterative manual play-throughs of VR scenes to finalize control flows(Ali et al., 2023). Given that VR product requirements are usually articulated in natural language (English), it can result in design revisions and necessitate multiple iterations for a single use case. Ultimately, developers select a design

[a] https://orcid.org/0009-0004-3372-6438
[b] https://orcid.org/0000-0001-7751-6070
[c] https://orcid.org/0000-0003-2280-5400

Jahagirdar, S., Karre, S. A. and Reddy, Y. R.
VReqDV: Model Based Design Generation & Design Versioning Tool for Virtual Reality Product Development.
DOI: 10.5220/0013433400003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 715-722
ISBN: 978-989-758-742-9; ISSN: 2184-4895

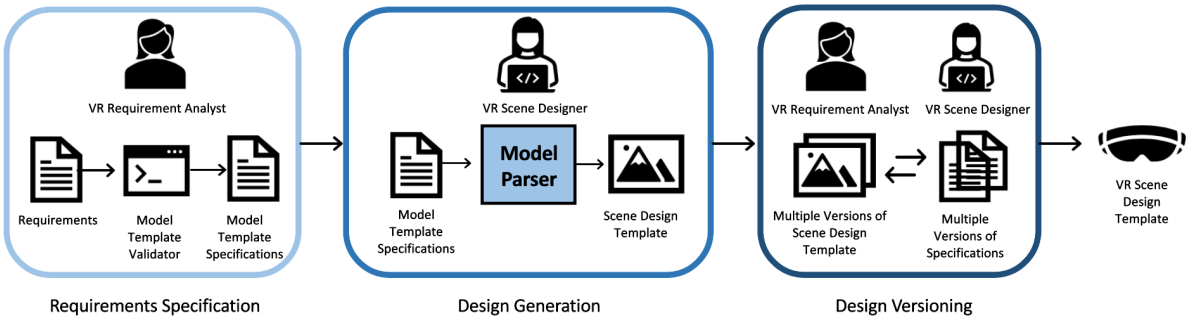

Figure 1: VR Requirements to Design Workflow.

template for further development after extensive modifications (Krauß et al., 2021). This iterative process generates design waste, with unacknowledged effort and creativity from designers going unrecognized. These observations clearly raise the following question: ***Is it possible to automatically generate design templates for VR products and maintain design versioning at the same time?.***

Given these observations, it is evident that an end-to-end automated design pipeline that supports automatic generation of design templates is a potential need. In this paper, we introduce **VReqDV**, a novel tool aimed at automating design template generation and supports version control. The tool leverages structured requirements specified in a prescribed format using our existing VR requirement specification tool called **VReqST** (Karre and Reddy, 2024), which is developed on the basis of a bare minimum meta-model of VR technology domain (Karre et al., 2023). By leveraging VReqST-based specifications, VReqDV aims to streamline design template generation and improve design versioning practices in VR product development.

The rest of the paper is structured as follows: In section 2, we present related work. In section 3, we provide the rationale for using VR meta-model, steps to specify requirements using VReqST, the proposed VR design workflow, and the complete model-driven VR designing pipeline. We illustrate the details of VR design generation and design versioning pipeline through VReqDV in sections 4 and 5. We detail some of the limitations and future work in our work in section 6.

2 RELATED WORK

Virtual Reality (VR) design generation has evolved in the past decade. Early studies on design generation focused on ontology-based approaches (De Troyer et al., 2003) (Geiger et al., 2000b), utilizing customized domain-specific semantics (De Troyer et al., 2007a) . These initial efforts were largely confined to specific domains such as e-commerce and recreational VR content. In the recent past, researchers developed visual semantic approaches for virtual scene generation (Zeng, 2011), which were later extended to text-to-3D scene generation using game-based learning content (Gradinaru et al., 2023). Most of these methods are now obsolete due to the advent of new game engines that are more robust and extensible for code generation. Baiqiang et al. (Gan and Xia, 2020) used VR as an intervention for conducting user experience evaluation through automation. Mengyu et al.(Chen et al., 2021) introduced a visual programming interface for VR scene generation, focusing on creating entanglement-based virtual objects. While both methods offer novel tools for designers, they primarily facilitate the foundational stages of VR environment setup and lack features for implementing custom behaviors and do not support design versioning. Recent work on Text-to-Metaverse (Elhagry, 2023) and Text2Scene (Tan et al., 2019) employ Generative Adversarial Networks (GAN) and Non-GAN based techniques to create VR environment. However, these methods do not provide mechanisms for designers to manipulate the generated scenes, nor do they incorporate essential design versioning features. VRGit (Zhang et al., 2023) is a step forward in VR design version control. It allows VR content creators to engage in real-time collaboration with design visualization and version tracking. However, VRGit does not offer automatic scene generation nor facilitate requirement conformance, limiting its application scope primarily to design versioning. Our work introduces a model-driven approach designed to integrate automatic scene generation, custom behavior implementation, and design versioning.

3 REQUIREMENTS TO DESIGN

A design pipeline represents a structured process intended to enhance design efficiency, ensure consistency, facilitate collaboration and increase productivity within the product development cycle. In case of VR product development, the design pipelines are limited to a few domain-centered applications and not well defined for overall VR product development (Ali et al., 2023). In this section, we outline the workflow for moving from requirements to design for VR products, one that incorporates a design pipeline consisting of design generation and design versioning.

VR Meta-Model: Researchers have used conceptual models to describe VR as a technology domain (De Troyer et al., 2007b)(Trescak et al., 2010). Karre et al. introduced a bare minimum meta-model template to illustrate the VR technology domain using a role-based modeling approach (Karre et al., 2023). This template captures the essential attributes required for building a VR software system and is extensible to domain-specific and application-specific VR elements. VReqDV uses this meta-model as a foundation to understand the requirements and convert them to design artifacts.

VReqST - Requirement Specification: VReqST can be used to specify requirements using an underlying meta-model of the VR technology domain (Karre et al., 2023). The meta-model contains bare minimum properties of a VR scene organized in model template files. It has five parts and each part is separately specified to complete the total specification of the VR application.

1. **Scene:** Defines the environment's spatial layout, terrain, and contextual settings.
2. **Asset:** Specifies objects within the scene, including their attributes and functionalities.
3. **Action-Responses:** Details user interactions and object behaviors within the VR environment.
4. **Custom Behavior:** Enables definition of non-standard interactions for enhanced flexibility.
5. **Timeline:** Organizes and synchronizes events, animations, and user interactions in sequence.

The model template files are represented in JavaScript Object Notation (JSON) format, which provides a flexible, text-based representation of the VR environment.

VReqDV Requirements to Design Workflow: A requirements to design workflow (as illustrated in Figure 1) can be established to programmatically generate VR design templates using requirements specified using VReqST. The key steps in the workflow are:

- *Requirements Elicitation:* The Requirements Analyst elicits requirements from the respective stakeholders to author the specifications using the VReqST tool.
- *Model Template Specifications:* Detailed specifications of the scene, assets (or) objects in the scene, action-responses associated with the objects in the scene, custom behaviors, and timeline of synchronous/asynchronous are published as specification model template files.
- *Design Generation:* The VR Scene Designer utilizes the specification model template files as input to the **Model Parser** component of VReqDV. The Parser extracts scene properties, object attributes, action-responses, custom behaviors, and timeline data from these JSON-format templates, converting them into UNITY-compatible specifications. This parsed output is then used to populate and generate an editable VR scene design template.
- *Editing Designs:* The VR Scene Designer can alter the generated design templates with desired changes and can save, revise, and publish new versions of the design templates.
- *Design to Specifications:* The saved design template version generates a corresponding requirement specification model template, ensuring traceability and synchronization between design iterations and their underlying specifications.
- *Finalising Design:* Finally, the VR Scene Developer uses the finalized design template to build the VR scene prototype.

4 VReqDV DESIGN GENERATION PIPELINE

Figure 3 illustrates the proposed design pipeline using our VR design generation and versioning tool called VReqDV. Following are the two major contributions of the proposed VR design pipeline:

1. Generate VR mock-up design templates using VReqST-based requirement model template specifications as input.
2. Support design versioning by provisioning VR Scene Designers to update & save the design templates with backward compatibility to propagate the saved changes to the requirement specifications.

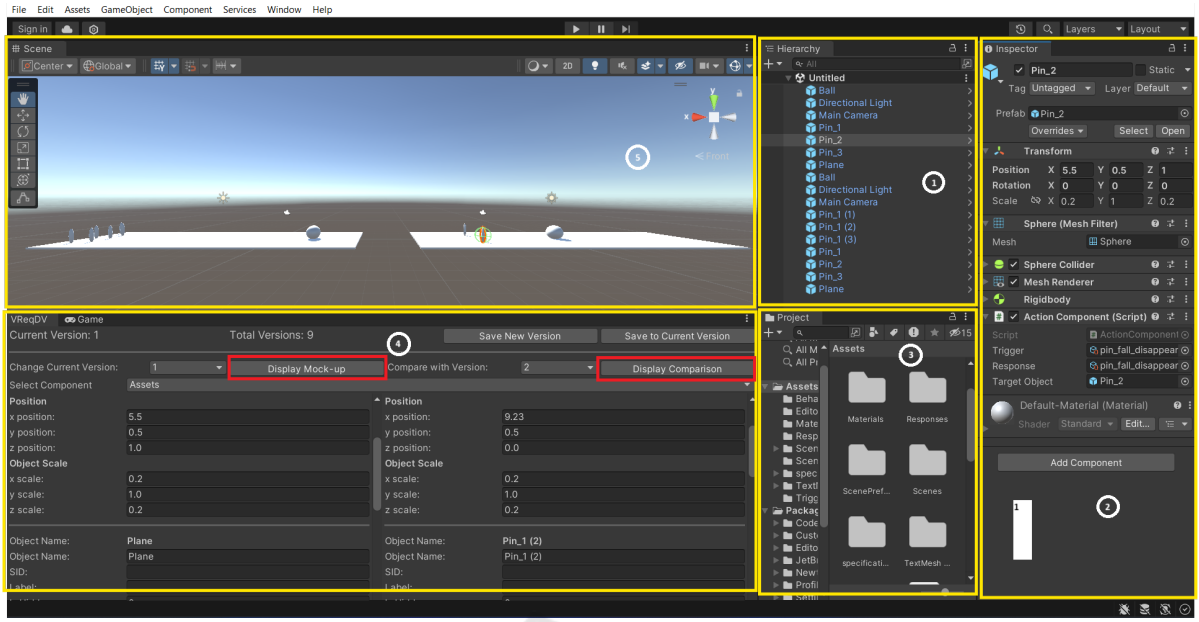

Figure 2: VReqDV Editor as Custom Window within the UNITY Game Engine.

As proof of concept, VReqDV has been implemented as a plugin to UNITY Game Engine [1]. Figure 2 illustrates the VReqDV editor in UNITY as a custom window. In the figure, the block **1** (top center) is the asset pane is the asset pane, which lists the available assets (or objects) in the VReqDV editor. Block **2** (right most) is the inspector window, displaying the properties of the selected object. Block **3** (bottom center) is the developer pane, listing the script files used to convert specifications into design templates. Block **4** (bottom left) is the specification pane, presenting detailed requirement specifications with comparison options. Block **5** (top left) is the design template pane, which shows designs for different specification versions; any revisions made will reflect in this pane. The highlighted boxes in red are buttons for generating design templates and for comparing design versions.

The VReqDV design generation pipeline consists of the sequence of scripts implemented in the Model Parser. We elaborate each stage of the pipeline (shown in Figure 3) and demonstrate an example using the VReqDV editor.

4.1 Scene Generation

The model parser processes scene specific aspects from the requirements specification to generate the initial virtual environment (shown in Figure 3). The scene model template includes essential elements such as scene identifiers, labels, play area dimensions, camera and viewport settings, clipping planes, and user interaction parameters. The parser generates a foundational scene template that supports subsequent attribute modifications.

4.2 Object Generation

After generating the initial layout from the scene specifications, the next phase involves adding objects to the virtual reality (VR) scene, as depicted in Figure 3. Each object specification includes various attributes, such as initial position, rotation, scale, light emission properties, shadow characteristics, gravity constants, and audio specifications.

The parser sequentially processes each object, initiating the creation of a basic geometric shape that corresponds to its designated primitive (e.g., cube, sphere, cuboid). The basic shape can then be modified based on the defined attributes. The refined objects are then instantiated in the scene, positioned and oriented according to their specifications. The current version of our work supports basic geometric shapes like cubes, spheres, and cuboids. To implement more complex geometries, multi-point polygons can be designed externally using software such as Blender[2] and then imported as part of the specifications.

To illustrate the concept, we present a simple example of a Bowling Alley VR scene, which includes three basic elements: a rectangular bowling lane, a

[1]https://unity.com/

[2]https://www.blender.org/

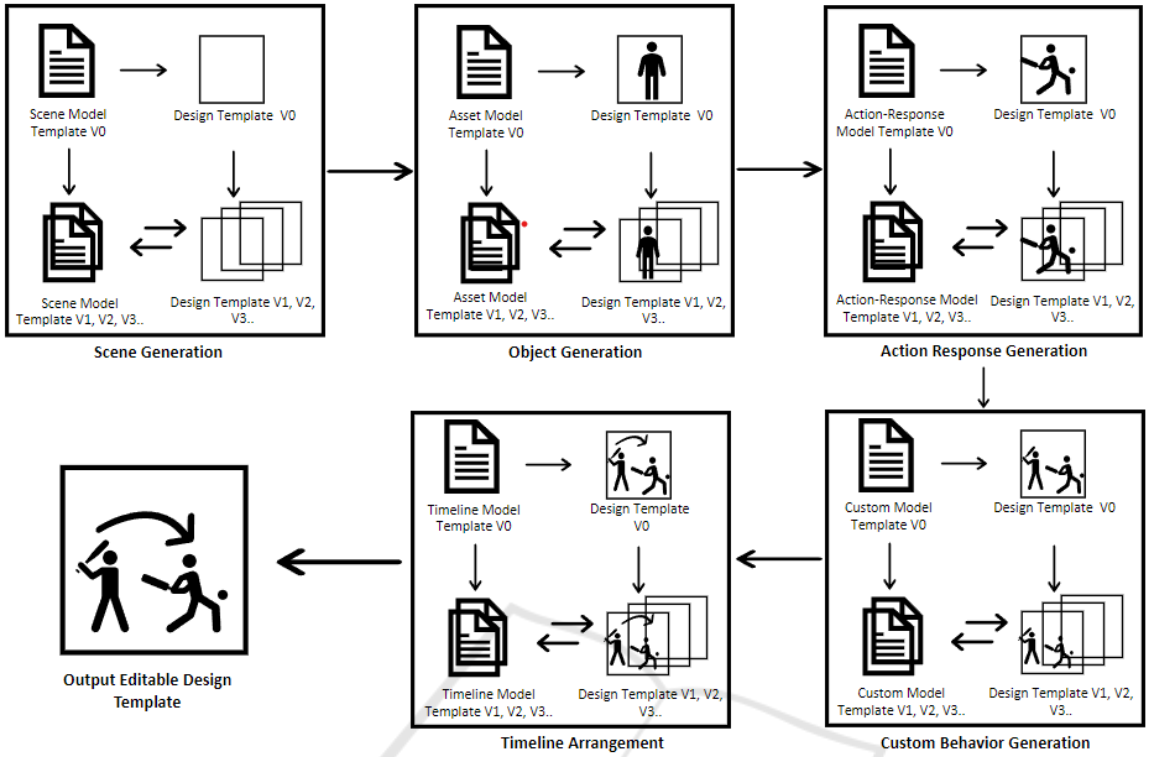

Figure 3: VReqDV Design Pipeline.

ball, and a single pin. Initially, all objects are static. Each object has specific properties; for instance, the ball is a "sphere," and the requirements analyst can set the IsIlluminate property to "true" within the specifications. Listing 1 shows a partial excerpt of the specifications used to establish the initial state of these objects.

Figure 4(a) displays the generated scene with the static objects described above. The complete specifications for these objects are available in the resource files (VReqDV, 2024).

4.3 Action-Response Generation

In this section, we discuss the development of dynamic properties for the generated objects. We define an **action-response** as a dual-event interaction involving a trigger event and a corresponding response event occurring between objects. Multiple objects can initiate known response events, regardless of their similarities. Within the VReqDV framework, we utilize UNITY's scriptable object feature to create modular and reusable components for triggers and responses, based on the foundational action-response specification model template.

To achieve this, we create a base class for each trigger and response type, along with specific predefined behavior templates. Possible trigger events include property changes, collisions, user inputs, temporal events, and audio alterations. Response events may involve actions such as movement, disappearance, user stimuli, and object instantiation. This list

```
"articles": [
    {
        "_objectname": "Ball",
        "shape": "sphere",
        "IsIlluminate": true,
        "Transform_initialpos": {
            "x": "0",
            "y": "0.5",
            "z": "0"
        },
        "Transform_initialrotation"
            : {
            "x": "0",
            "y": "0",
            "z": "0"
        },
        "Transform_objectscale": {
            "x": "1",
            "y": "1",
            "z": "1"
        },
        ...
```

Listing 1: Object specification: Ball

will evolve over time. When setting up scene interactions, we instantiate trigger and response events from these templates, adjusting them based on input specifications. The algorithm for generating action-response behaviors is described in Algorithm 1.

Algorithm 1: Adding Behavior to an Object.

Input: Specifications: (trigger, response)
Output: Behavior added to the source object
begin
  Step 1: Identify predefined trigger template for the trigger event given.
  Step 2: Initialize instance of trigger template.
  Step 3: Configure the parameters based on input specifications.
  Step 4: Repeat steps 1–3 for response event.
  Step 5: Initialize an instance of Action Component and set its variables with the configured trigger and response.
  Step 6: Add Action Component as behavior of the source object.
end

Figure 4 illustrates the action-response generated for the bowling alley game scene where the Pin falls (Figure 4(c)) upon collision with the Ball (Figure 4(b)). Alternatively, we can define an action-response for the Pin, so it disappears upon falling. The trigger event is the tilting, and the response is the Pin's disappearance.

Here, the trigger corresponding to change in object rotation will be instantiated. The input directs the trigger to set a threshold of 10 degrees for change in rotation of the Pin. Consider Figure 4(e), where upon falling, the response is initiated from the template corresponding to object visibility, and set to disappearing as shown in Figure 4 (f). The *actioncomponent* here is assembled with these behaviors to generate the action flow in Figure 4 (e) and Figure 4 (f). Overall, VReqDV will associate these action-responses templates as components of the source object, that links triggers to responses and when executed. As part of current implementation for the provided example, the dynamic behavior using its custom specification (Figure 3: Custom Behavior Generation) doesn't apply. Thus we excluded it as part of the overall illustration.

4.4 Scene Timeline

The timeline feature expands upon the action-response concept by providing VR scene designers with the ability to order response events as required and define both synchronous and asynchronous events. This adds flexibility for design and allows more complex and nuanced scene interactions, enhancing the overall user experience and creative potential of the system. In the current edition of VReqDV, the underlying timeline can be modelled exhaustively as an input structure by effectively describing action-response sequences. There is scope for improvement for implementing the timeline onto the scene design in the UNITY editor without the dependency on an action-response sequence.

5 DESIGN VERSIONING

VReqDV's version control system uses VReqST specification files as the base to create a repository of VR scene versions. Scene properties, objects, their properties, and dynamic behaviors are captured from the scene and converted to textual representation in model template files (JSON). This eliminates the need to store multiple specification files associated with large VR scenes files. VReqDV utilizes a versioning mechanism through a reverse-engineered Model Parser, allowing the conversion of UNITY scenes into model template specification files based on the VReqST framework. When scene templates are modified, they can be saved as new versions, enabling version tracking and change management. This facilitates scene differencing, verification against initial requirements, and analysis, similar to functionalities of version control systems like Git. Figure 5 shows a side-by-side comparison of two versions of action-response specifications for a bowling alley scene, where the pin disappears upon falling.

VReqDV can also convert model template specifications back into scene representations, providing intuitive visual differentiation between versions. This ensures that properties and state information related to objects are stored in new model template version files.

To compare scene versions, designers select version numbers in the 'compare versions window,' allowing VReqDV to load both versions side-by-side for easy identification of changes, as shown in Figure 2. This versioning system enables continuous requirements validation, simplifies version comparison, and accelerates the iterative design process.

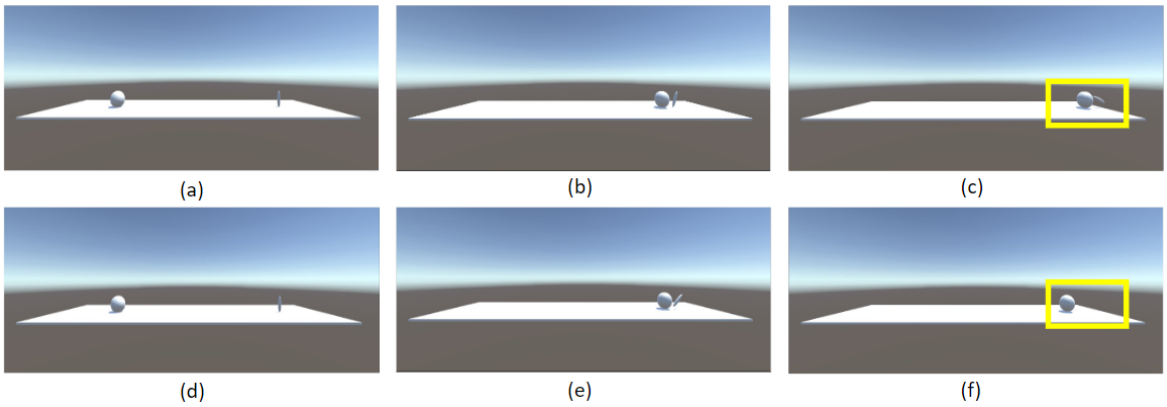

Figure 4: Two versions of Action Response - (a)-(c) Ball collides with the Pin, Pin falls of the ground, (d)-(f) Ball collides with the Pin, Pin falls and disappears.

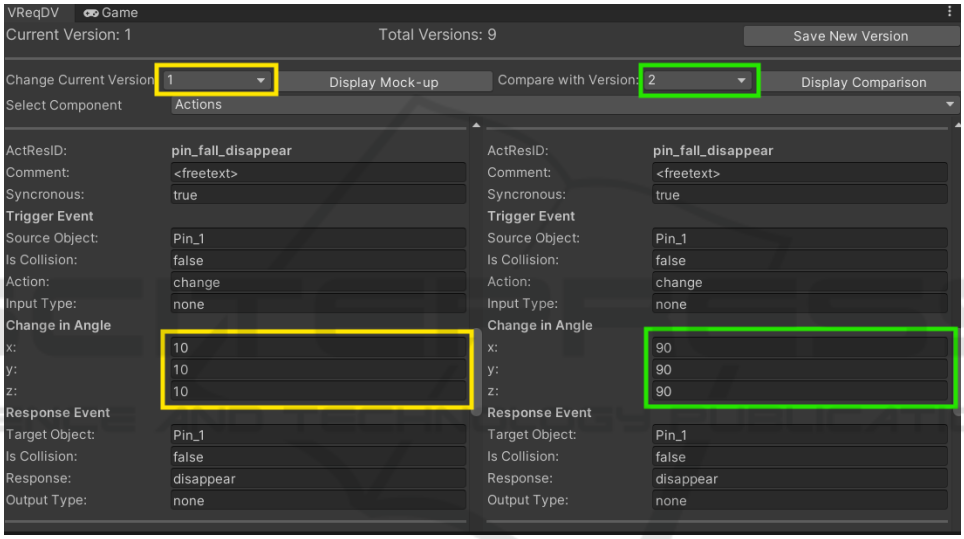

Figure 5: Action responses compared between versions: Pin falling behavior.

6 LIMITATIONS & FUTURE WORK

VReqDV is designed to standardize the VR product development pipeline, accepting only requirements authored with VReqST, a specification tool backed by a VR technology meta-model. It utilizes predefined scriptable objects to generate dynamic properties for virtual objects, differing from traditional manual programming, which can limit flexibility and handle complex interactions. This reliance on predefined actions may oversimplify behaviors and requires developers to create customized behaviors, potentially increasing development time.

The lack of automated code generation complicates unit testing and debugging, making it harder to trace behavior errors. This could be improved if VReqDV generates code from design templates, enhancing the reliability of VR environments.

As VR design scenarios grow, the number of predefined scriptable objects may increase, necessitating optimized design strategies to prevent confusion. Currently, VReqDV is tailored for the UNITY Game Engine but can adapt to other VR SDKs like Unreal Engine and Blender.

While demonstrated through a simple bowling alley example, extensive validation studies and open-source plugins for various SDKs are planned for the future. Future efforts will focus on extending VReqDV with comprehensive version control, advanced visualization tools, and strategies for integrating code generation and automated testing, aiming to streamline VR development and enable more complex applications.

REFERENCES

Ali, M. A. M., Ahmad, M. N., Ismail, W. S. W., Aun, N. S. M., Basri, M. A. F. A., Fazree, S. D. M., and Zakaria, N. H. (2023). A virtual reality development methodology: A review. In *Advances in Visual Informatics: 8th International Visual Informatics Conference, IVIC 2023, Selangor, Malaysia, November 15–17, 2023, Proceedings*, page 26–39, Berlin, Heidelberg. Springer-Verlag.

Balzerkiewitz, H.-P. and Stechert, C. (2021). Product development methods in virtual reality. *Proceedings of the Design Society*, 1:2449–2458.

Cao, Y., Ng, G.-W., and Ye, S.-S. (2023). Design and evaluation for immersive virtual reality learning environment: A systematic literature review. *Sustainability*, 15:1964.

Chen, M., Peljhan, M., and Sra, M. (2021). Entanglevr: A visual programming interface for virtual reality interactive scene generation. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21, New York, NY, USA. Association for Computing Machinery.

De Troyer, O., Bille, W., Romero, R., and Stuer, P. (2003). On generating virtual worlds from domain ontologies. In *MMM*, pages 279–294.

De Troyer, O., Kleinermann, F., Mansouri, H., Pellens, B., Bille, W., and Fomenko, V. (2007a). Developing semantic vr-shops for e-commerce. *Virtual Reality*, 11:89–106.

De Troyer, O., Kleinermann, F., Pellens, B., and Bille, W. (2007b). Conceptual modeling for virtual reality. In Grundy, J., Hartmann, S., Laender, A. H. F., Maciaszek, L., and Roddick, J. F., editors, *Tutorials, posters, panels and industrial contributions at the 26th International Conference on Conceptual Modeling - ER 2007*, volume 83 of *CRPIT*, pages 3–18, Auckland, New Zealand. ACS.

Elhagry, A. (2023). Text-to-metaverse: Towards a digital twin-enabled multimodal conditional generative metaverse. In *Proceedings of the 31st ACM International Conference on Multimedia*, MM '23, page 9336–9339, New York, NY, USA. Association for Computing Machinery.

Gan, B. and Xia, P. (2020). Research on automatic generation method of virtual reality scene and design of user experience evaluation. In *2020 International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*, pages 308–312.

Geiger, C., Paelke, V., Reimann, C., and Rosenbach, W. (2000a). A framework for the structured design of vr/ar content. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '00, page 75–82, New York, NY, USA. Association for Computing Machinery.

Geiger, C., Paelke, V., Reimann, C., and Rosenbach, W. (2000b). A framework for the structured design of vr/ar content. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 75–82.

Gradinaru, A., Vrejoiu, M., Moldoveanu, F., and Balutoiu, M.-A. (2023). Evaluating the usage of text to3d scene generation methods in game-based learning. In *2023 24th International Conference on Control Systems and Computer Science (CSCS)*, pages 633–640.

Karre, S. A., Mathur, N., and Reddy, Y. R. (2019). Is virtual reality product development different? an empirical study on vr product development practices. In *Proceedings of the 12th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ISEC '19, New York, NY, USA. Association for Computing Machinery.

Karre, S. A., Pareek, V., Mittal, R., and Reddy, R. (2023). A role based model template for specifying virtual reality software. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

Karre, S. A. and Reddy, Y. R. (2024). Model-based approach for specifying requirements of virtual reality software products. *Frontiers in Virtual Reality*, 5.

Krauß, V., Boden, A., Oppermann, L., and Reiners, R. (2021). Current practices, challenges, and design implications for collaborative ar/vr application development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA. Association for Computing Machinery.

Mattioli, F., Caetano, D., Cardoso, A. M. K., and Lamounier, E. A. (2015). On the agile development of virtual reality systems.

Tan, F., Feng, S., and Ordonez, V. (2019). Text2scene: Generating compositional scenes from textual descriptions. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6703–6712.

Trescak, T., Esteva, M., and Rodriguez, I. (2010). A virtual world grammar for automatic generation of virtual worlds. *Vis. Comput.*, 26(6–8):521–531.

Troyer, O. D., Bille, W., and Kleinermann, F. (2009). Defining the semantics of conceptual modeling concepts for 3d complex objects in virtual reality. *J. Data Semant.*, 14:1–36.

VReqDV (2024). Vreqdv. https://github.com/VreqDV/vreqdv-tool.

Wu, H., Cai, T., Liu, Y., Luo, D., and Zhang, Z. (2021). Design and development of an immersive virtual reality news application: a case study of the sars event. *Multimedia Tools Appl.*, 80(2):2773–2796.

Zeng, X. (2011). Visual semantic approach for virtual scene generation. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '11, page 553–556, New York, NY, USA. Association for Computing Machinery.

Zhang, L., Agrawal, A., Oney, S., and Guo, A. (2023). Vrgit: A version control system for collaborative content creation in virtual reality. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery.