A Programming Model & Architecture for Adaptive Virtual Reality Training

Kristof Overdulve, Ruben Simons, Lowie Spriet and Yasmine Wauthier
Immersive Lab, AP University of Applied Sciences and Arts
Antwerp, Belgium
Email: {firstname}.{lastname}@ap.be

***Abstract*—Virtual Reality (VR) is an effective medium for developing and practicing skills. Combined with instructional design, it can potentially have a more significant effect on learning efficiency, learning curve, and retention than classical lectures or instruction videos. As a result, the amount of VR training has exploded. Building convincing and interactive VR training experiences is expensive. Even after considerable investment, the result is often a static learning experience that does not adjust to the user's needs and skill set and offers a small number of training scenarios with no variations to improve replay value.**

Therefore, we propose a programming model & architecture for adaptive VR training experiences. From a deployment standpoint, it works by launching VR training experiences from training configuration files. These files describe training scenarios from a well-defined semantic structure. Such an architecture essentially decouples the problem of generating training scenarios, which developers can do without game development experience, from the challenge of creating real-time, immersive VR experiences. End-users can generate the training configuration files through a user-friendly training scenario editor. Software developers can also use Artificial Intelligence algorithms to procedurally generate new training scenarios which adapt automatically to the user's skill set.

Our programming model generalizes well to novel VR training types and has been applied successfully to fire training, evacuation training and the manipulation of industrial machinery.

***Index Terms*—Virtual Reality, adaptive VR training**

I. Introduction

Virtual Reality (VR) has gained significant popularity since introducing high-quality, affordable head-mounted displays such as the HTC Vive and the Oculus Quest. It offers an experience in which a three-dimensional environment or image is shown on the VR display with a wide field of view, allowing the headset's wearer to feel wholly submersed in the virtual world. It offers a compelling experience, allowing users to feel as if they were in the virtual world instead of the physical world in which they are during the experience.

VR can not only be used for entertainment purposes but also for productive and creative applications [1]. Due to its convincing depiction of reality, it is an effective medium for developing and practicing skills [2]. Combined with instructional design, it can potentially have a more significant effect on learning efficiency, learning curve, and retention than classical lectures or instruction videos [3]. As a result, the amount of VR training simulations has grown exponentially [4], [5]. Nevertheless, they suffer from several drawbacks limiting their effectiveness.

- The controls through VR controllers seldom accurately mimic the motor patterns the actual movements require.
- Depending on the artistic skills of the authors of the simulations, the environments looks unrealistic.
- The training is often a static learning experience that is not tailored to the userś skill set and offers little replay value after the training has been completed [6].

Building VR training experiences is expensive and time-consuming [7]. To increase the return on investment, there is an urgent need for adaptive VR training experiences which allow cost-effectively taking building blocks of a VR training experience and using them to create a wide variety of scenarios [8]–[10]. Educational designers can then create new training scenarios to offer an extensive training program challenging the prospective students with a multitude of learning goals at their skill levels. Additionally, Artificial Intelligence (AI) algorithms could automatically analyze the students' skill level to make training scenarios more challenging, reorder the training scenarios or offer in-time feedback. Such adaptive VR training experiences have a superior replay value, a higher learning efficiency [11], [12], and a better return on investment than static learning experiences.

This paper presents a programming model and software architecture through which software developers can build adaptive VR training experiences. Our programming model generalizes well to novel VR training scenarios and has been applied successfully to fire training, evacuation training, and the manipulation of industrial machinery.

II. Related work

Our work is inspired by tools such as Zapier[1] and IFTTT[2], allowing end-users with no programming knowledge to integrate multiple apps. They do this by defining actions in reaction to specific triggers. An example integration is "*(trigger)* When I get a new lead on Facebook, *(action)* add the lead to my MailChimp mailing list.". These systems are similar to our vision on adaptive VR training as they also focus on allowing end-users to extend the power of existing components

This research was funded by VLAIO as a TETRA project, "AI-driven VR training in an adaptive user context."

[1]https://www.zapier.com
[2]https://ifttt.com/

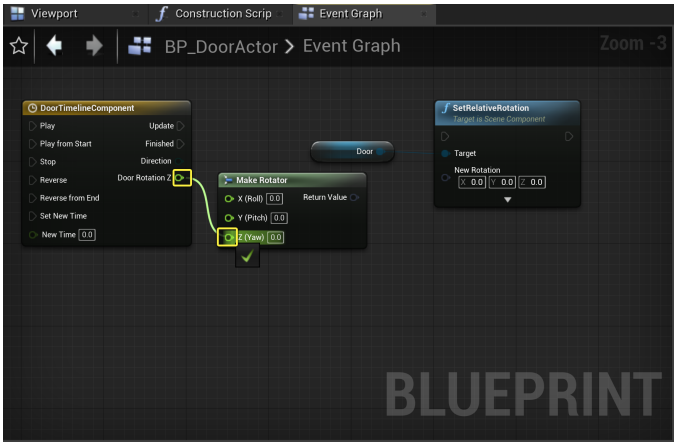

Fig. 1. A screenshot of an Unreal Blueprint graph defining the behavior of opening a door. This image is best viewed on a screen.

without requiring programming knowledge. Similar to our programming model, these tools require carefully designing the allowed events and reactions to be useful for end-users without technical expertise. The field of reactive programming [13] and frameworks such as MobX[3] also inspire our work, allowing listening to changes of defined properties and reactions to those changes.

Our work is also related to the Blueprints Visual Scripting system in Unreal Engine[4], shown in figure 1. It is a graphical scripting system to build interactive games. In Unreal Blueprints, gameplay elements are defined as nodes and linked through a wide variety of events. However, as Unreal Blueprints lets game designers build logic starting from low-level 3D models, it is too complex and therefore unsuitable for non-technical end-users. Our system operates on a much higher level of abstraction by turning high-level game objects into adaptive objects with high-level interactions.

Adaptive VR training is not new. Its efficiency and benefits have repeatedly been demonstrated in various domains [8]. Prototypes have been built to create personalized experiences to optimally help users reach the intended learning goals [10], [14]–[20]. Although earlier work demonstrates the benefits of adaptive VR training and details algorithms that customize the experience, they provide very few details on how software developers and VR agencies can actually *build* adaptive VR training experiences. On the other hand, our work focuses on the complementary and non-trivial problem of creating a programming model & architecture through which software developers and VR agencies can build generic adaptive VR training experiences. Through this effort, we hope to broaden the impact of adaptive VR training.

[3]https://mobx.js.org/

[4]https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/

III. Background information

We integrated our adaptive VR training framework with the Unity[5] game engine. Although our programming model is easily transferrable to other game engines, this section gives a brief overview of how Unity works and the concepts needed to understand how our programming model works on top of it.

Unity follows a component-based software design. *Game objects* are the fundamental objects in Unity to represent characters, light, 3D primitives, and so on. They act as a container for *components* that implement the actual functionality of the game objects. Unity contains a library of built-in components to determine game objects' positions, draw 3D primitives, lights, and so on. Custom components are built through C# scripts. Game objects and components depict a low-level interface intended for game designers and developers. It is often more convenient to reason about high-level entities, such as a fire, encapsulating 3D primitives, animations, behavior and interaction. *Prefabs* in Unity help in this regard. They turn a game object with specific components and properties into a high-level template that can be reused and instantiated multiple times. Our programming model uses the prefab system to instantiate high-level game objects and extend them with an adaptive layer to add configurability for adaptive VR training experiences.

IV. System overview

We first detail how our system for adaptive VR training experiences works from a deployment point of view, we then give an overview of the different high-level components in the architecture, and how they interact. We then detail the programming model SDK, the runtime implementation, and conclude with how the reader can make their VR training adaptive as a set of concrete steps.

A. Deployment architecture

The proposed architecture effectively decouples the process of generating training scenarios from actually launching them as a VR training experience, as depicted in figure 2. Different types of processes can thus generate JSON data consumed by our *Adaptive Training Runtime* in Unity: web-based GUIs to allow end-user editing, AI algorithms to procedurally generate new training scenarios, personalize training scenarios to individual users, provide feedback, and so on. We used JSON as a serialization format. The JSON training scenario could come from a REST backend, or be packaged within the Unity app as a file.

B. Component overview

The components of our adaptive VR training Software Development Kit (SDK) are depicted in figure 3. They consist of two reusable components: the Adaptive Training Application Programming Interface (API) described in section IV-C and the Adaptive Training Runtime environment detailed in section

[5]https://unity.com/

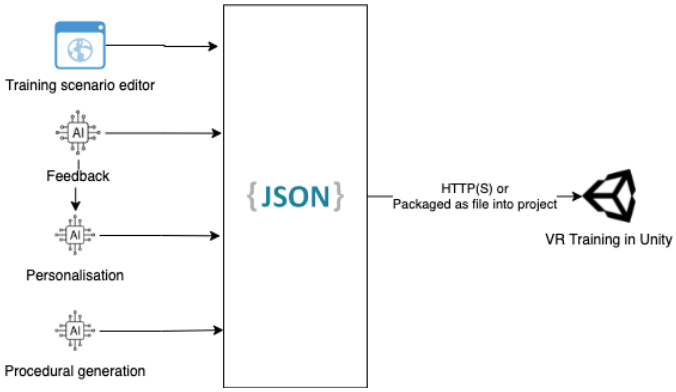

Fig. 2. An illustration of the proposed deployment architecture. Any external process can turn semantically correct JSON training scenarios into an actual training experience. Here we show examples of a training editor with a GUI and several AI algorithms.

IV-D responsible for running the adaptive layer. The use of the SDK is detailed in IV-E. When developing a concrete training application, a VR training Unity project with a set of reusable prefab objects needs to be created, and a collection of Adaptive Entity Definitions needs to be defined and attached to the prefabs. Aside from connecting Adaptive Entity Definitions to prefabs, they define contracts to assist in generating training scenarios.

C. Programming model

This section depicts the semantic structure of the scenario configuration file and details the Adaptive Training API.

The scenario configuration file follows a graph-like model where vertices are Adaptive Entities, and the edges define reactions on property changes of these entities. A simple scenario defining an adaptive fire, a person, and connecting them is shown visually in figure 4 and as JSON in appendix A.

The adaptive entities, represented as vertices in the graph, are high-level game objects with well-defined behavior, and a high-level collection of adaptive properties. Crucial in the success of the proposed programming model is that game developers only expose high-level entities with concrete adaptive properties. Only the properties that differ between training scenarios and the reactions to those adaptive properties should be exposed. In addition to this, only by exposing exclusively high-level interfaces can one expect end-users without technical skills and AI algorithms to create scenarios. Good examples of adaptive properties are properties that make a scenario less or more complex and high-level properties such as the intensity of a fire, whether a person panics or not, whether a door is open or not, and so on. Less effective adaptive interfaces include low-level properties (e.g., defining an animation opening the door from a specific closed angle to an open angle around a quaternion, the particles of a fire) and static behavior that does not change between scenarios (e.g., when a lever is manipulated, an alarm signal should sound). The Adaptive Entities are transformed by the Adaptive Training Runtime to Unity game objects when the scene loads and initialized with the JSON data.

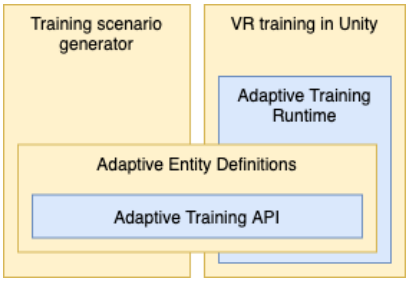

Fig. 3. The components involved in an adaptive VR training experience. The reusable Adaptive Training API and Runtime engine are shown in blue. The Adaptive Training API is used to define Adaptive Entities which are deployed in Unity through the Adaptive Training Runtime. Additionally, external processes can use the adaptive entities to generate training scenarios to detect which properties are adaptive.

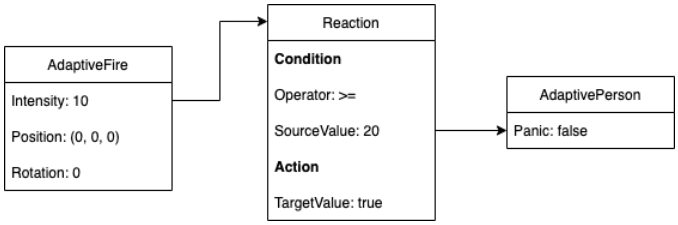

Fig. 4. A simple example of a scenario in which a person reacts after detecting fire. The adaptive entities are AdaptivePerson and AdaptiveFire. Fire can have a certain intensity, and the person has an adaptive state of panic. Once the fire reaches an intensity of 10, the person panics.

A "reaction" edge links two properties of Adaptive Entities: an incoming property and an outgoing one. The edge listens to the incoming property for changes in its value. If changed, the edges evaluate a given condition and runs an action on the outgoing property if the condition holds. The action changes the property value of the outgoing property to a new value. In our concrete example depicted in figure 4, the reaction edge has the intensity of a fire as the incoming property value. When the fire intensity is larger than or equal to the value of 20, the edge fires a property change on the person to change its panic property to true.

A reaction has the following parameters:

- **Operator:** when the incoming property has changed, compare it with the given operator. Operators can be = (equals), > (larger than), >= (larger than or equal to), < (smaller than), or <= (smaller than or equal to).
- **SourceValue:** the value to which the operator compares the incoming property.
- **TargetValue:** the value to which the target property changes when the condition is true.

The reaction edge waits for a property change of the incoming value. It then compares it to the `SourceValue` using the given operator. The action fires when the comparison holds. Upon firing the action, the outgoing property changes to the `TargetValue`.

```
using UnityEngine;

public class AdaptiveContainer :
    MonoBehaviour
{

    [SerializeField]
    private Object adaptiveScript;
}

[AdaptiveObject("Fire")]
public class AdaptiveFire : MonoBehaviour
{
    [AdaptiveProperty("intensity")]
    public float Intensity { get; set; }

}
```

Fig. 5. A simplified definition of the AdaptiveContainer class and an example third-party class using the Adaptive Training API to make a Fire prefab adaptive.

The public Adaptive Training API consists of the `AdaptiveContainer` class and the `AdaptiveObject` & `AdaptiveProperty` attributes. To make a high-level game object adaptive before turning it into a prefab, the `AdaptiveContainer` script needs to be attached to the root of the game object. The `adaptiveScript` parameter then needs to be initialized with a third-party class defining the adaptive features of the entity through C# attributes. The attributes are used to denote the class of the adaptive object in the JSON data and the name and type of adaptive properties. The public API is illustrated in figure 5.

Internally, the read JSON data in combination with the adaptive objects and properties are converted into a structure illustrated in figure 6 before being processed by the runtime environment.

This programming model is sufficiently powerful to handle many scenarios and sufficiently simple to allows AI algorithms and end-users to understand and generate the scenario files. A screenshot of an example graphical training scenario editor is displayed in figure 7.

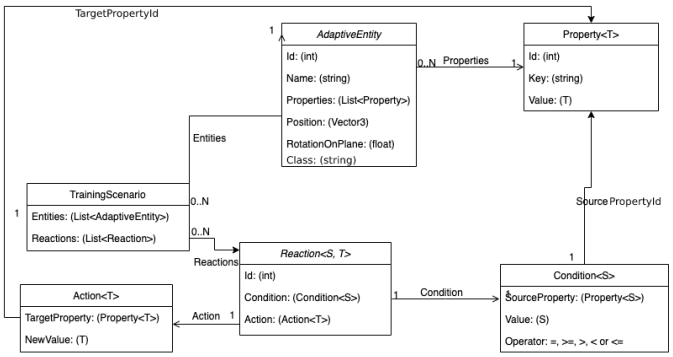

Fig. 6. The class diagram depicting the adaptive entities, reaction, and their relationship.

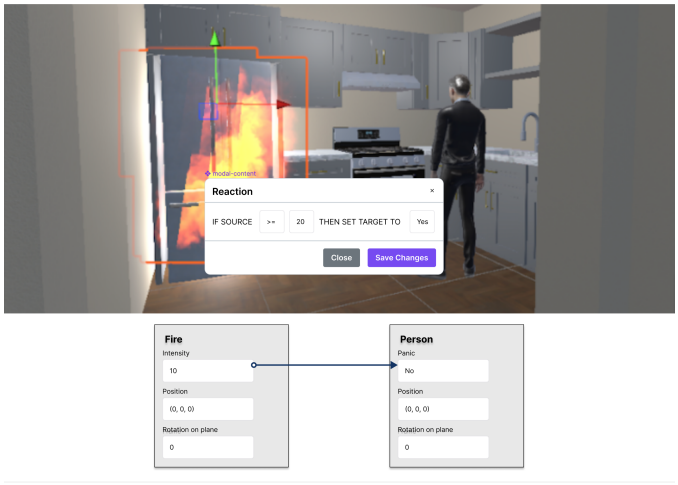

Fig. 7. A sample GUI through which end-users can generate new training scenarios. The data powering the interface comes from the Adaptive Object and Adaptive Property attributes and the defined prefabs.

D. Adaptive Training Runtime

The implementation details of the Adaptive Training Runtime framework driving the adaptive programming model are shown in figure 8

The `ScenarioMaster` of the runtime framework needs to be initialized when launching the scene. Upon initialization, it reads the training configuration file from the defined JSON input stream. This stream can be a file or a network stream. The scenario master makes an inventory of all available prefabs and lists the ones with the `AdaptiveObject` attribute to check their value. The scenario master then instantiates the prefabs, stores a reference to the instantiated objects, and initializes

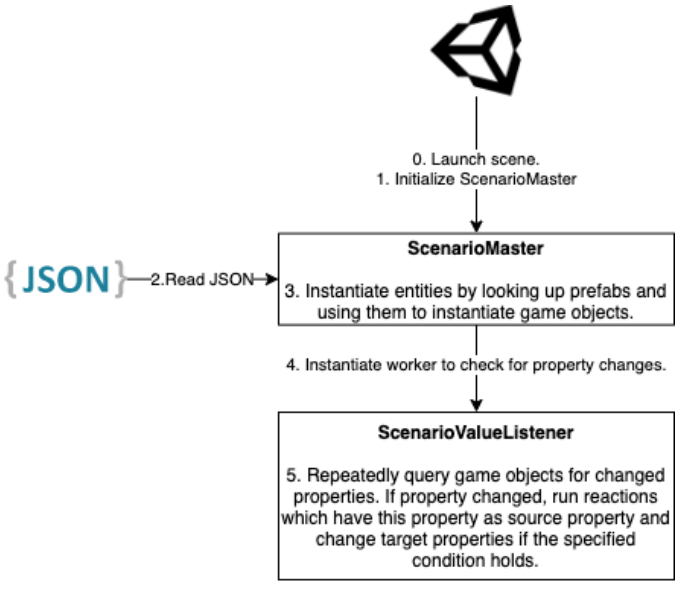

Fig. 8. The implementation details of our adaptive programming model. The input of this runtime environment is the collection of prefab objects and the JSON training configuration file.

Fig. 9. A screenshot of our fire & evacuation training.

Fig. 10. A screenshot of our industrial machinery training.

the adaptive properties with the values from the JSON data. The list of references to instantiated prefabs is used by the `ScenarioValueUpdater` worker to repeatedly check the instantiated prefabs for changes in one of the adaptive properties attached to the entity through adaptive objects. If a property changes, it performs the actions defined in the reaction(s) which have this property as incoming property. The choice for a centralized worker module to check for property changes, rather than giving this responsibility to adaptive objects, allows intelligent scheduling of the worker and alleviates the burden of requiring third-party developers to remember notifying the central system.

E. Development process

The development process of an adaptive VR training experience consists of two stages.

1) Firstly a game developer builds a collection of game objects to convert to high-level prefabs.
2) The game developer attaches an `AdaptiveContainer` to the game object to be made adaptive. We recommend keeping the adaptive and fixed properties separated and naming the script with adaptive properties `Adaptive<Prefab name>`. A simple `AdaptiveFire` script is depicted in 5. The properties annotated with the `AdaptiveProperty` attribute are customizable properties used by the Adaptive Training Runtime.

V. Applications

1) Fire & evacuation training: Our first application, displayed in figure 9, is a fire & evacuation training experience. It was built from a set of reusable components: fire, extinguishers, non-playable characters, and so on. The user performing the VR training can take the role of designated safety personnel, employee, or visitor of a building. We applied our adaptive VR training SDK successfully to implement a wide variety of evacuation & fire hazard scenarios covering multiple difficulty factors: adapting the intensity of fires, the location of fires, windows or doors being open or not, the presence of fire extinguishers, people panicking, and so on.

2) Industrial machinery training: Our second application, displayed in figure 10, is a VR training program for operating expensive and dangerous machinery found on industrial sites. Students used this environment as a virtual training ground to test and measure different components within the virtual simulation during and after the COVID-19 pandemic. The training simulates the physical equipment the students have available to them in the lab of the University. Learning and testing their skills in a virtual environment allows for more freedom to learn about hazardeous scenarios that cannot be simulated in real-life such as explosions, implosions, gas hazards.

Our adaptive framework has been applied to allow teachers to create a broad range of training scenarios where different types of errors must be solved and multiple tasks must be performed successfully. The training scenarios also specified multiple levels of complexity with customized support and feedback.

VI. Discussion

We believe our programming model is broadly applicable to a multitude of training scenarios. However, one could say that any programming model attempting to give editing control to designers and end-users will always end up as complicated as Unreal Blueprints. If you choose to expose low-level 3D meshes in an adaptive programming model, this point of view is indeed valid. Therefore, we emphasize the importance of only exposing the properties of high-level entities that require adaptability. Good examples of adaptive properties are the entities' position in the world, properties through which we can make the training scenario more difficult or more accessible, and places where the interaction between multiple entities depends on the training scenario. Using these guidelines, we believe our model allows end-users editing without devolving to a visual version of a programming language that only technical software developers understand.

Even though adaptive VR training experiences significantly improve the return on investment of VR training, developing for photorealism or complex interactivity remains exceptionally expensive. Especially in fields where the training requires realistic human interactions or photorealism. We see

a lot of potential in VR training using 360-degree video or Image-Based Rendering algorithms [21]. Such VR training experiences would require a mix of modeled 3D primitives and video-based imagery for complex interactions. Future work includes more cost-effectively obtaining VR content for training purposes.

VII. Conclusion

In this paper, we introduced a simple-to-understand and efficient programming model and software architecture for developing adaptive VR training experiences. This model has the potential to significantly improve the efficiency and replayability of VR training experiences by making them adaptive. The programming model has been shown to generalize well to novel VR training experiences such as evacuation & fire training, and manipulation of industrial machinery. We believe it is widely applicable beyond these types of training experiences. Nevertheless, to prove its applicability to a wide range of scenarios, we welcome VR developers to apply our programming model and propose enhancements where the model was not expressive enough to meet their needs.

References

[1] J. Bissonnette, F. Dubé, M. D. Provencher, and M. T. Moreno Sala, "Evolution of music performance anxiety and quality of performance during virtual reality exposure training," *Virtual Reality*, vol. 20, no. 1, pp. 71–81, 2016.

[2] V. N. Palter and T. P. Grantcharov, "Individualized deliberate practice on a virtual reality simulator improves technical performance of surgical novices in the operating room: a randomized controlled trial," 2014.

[3] D. M. Markowitz, R. Laha, B. P. Perone, R. D. Pea, and J. N. Bailenson, "Immersive virtual reality field trips facilitate learning about climate change," *Frontiers in psychology*, vol. 9, p. 2364, 2018.

[4] L. Freina and M. Ott, "A literature review on immersive virtual reality in education: state of the art and perspectives," in *The international scientific conference elearning and software for education*, vol. 1, no. 133, 2015, pp. 10–1007.

[5] J. Tichon and R. Burgess-Limerick, "A review of virtual reality as a medium for safety related training in mining," *Journal of Health & Safety Research & Practice*, vol. 3, no. 1, pp. 33–40, 2011.

[6] H. Engelbrecht, R. W. Lindeman, and S. Hoermann, "A swot analysis of the field of virtual reality for firefighter training," *Frontiers in Robotics and AI*, vol. 6, p. 101, 2019.

[7] "Walmart, fedex make case for vr training," https://www.techtarget.com/searchhrsoftware/news/252472139/Walmart-FedEx-make-case-for-VR-training, accessed: 2022-08-25.

[8] M. Zahabi and A. M. Abdul Razak, "Adaptive virtual reality-based training: a systematic literature review and framework," *Virtual Reality*, vol. 24, no. 4, pp. 725–752, 2020.

[9] C. Peretz, A. D. Korczyn, E. Shatil, V. Aharonson, S. Birnboim, and N. Giladi, "Computer-based, personalized cognitive training versus classical computer games: a randomized double-blind prospective trial of cognitive stimulation," *Neuroepidemiology*, vol. 36, no. 2, pp. 91–99, 2011.

[10] N. Vaughan, B. Gabrys, and V. N. Dubey, "An overview of self-adaptive technologies within virtual reality training," *Computer Science Review*, vol. 22, pp. 65–87, 2016.

[11] D. Bian, J. Wade, Z. Warren, and N. Sarkar, "Online engagement detection and task adaptation in a virtual reality based driving simulator for autism intervention," in *international conference on universal access in human-computer interaction*. Springer, 2016, pp. 538–547.

[12] J. D. Vermunt and N. Verloop, "Congruence and friction between learning and teaching," *Learning and instruction*, vol. 9, no. 3, pp. 257–280, 1999.

[13] E. Bainomugisha, A. L. Carreton, T. v. Cutsem, S. Mostinckx, and W. d. Meuter, "A survey on reactive programming," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–34, 2013.

[14] C. Baker and S. H. Fairclough, "Chapter 9 - adaptive virtual reality," in *Current Research in Neuroadaptive Technology*, S. H. Fairclough and T. O. Zander, Eds. Academic Press, 2022, pp. 159–176. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128214138000142

[15] K.-C. Siu, B. J. Best, J. W. Kim, D. Oleynikov, and F. E. Ritter, "Adaptive virtual reality training to optimize military medical skills acquisition and retention," *Military medicine*, vol. 181, no. suppl_5, pp. 214–220, 2016.

[16] M. Ma, M. McNeill, D. Charles, S. McDonough, J. Crosbie, L. Oliver, and C. McGoldrick, "Adaptive virtual reality games for rehabilitation of motor disorders," in *international conference on universal access in human-computer interaction*. Springer, 2007, pp. 681–690.

[17] N. Rossol, I. Cheng, W. F. Bischof, and A. Basu, "A framework for adaptive training and games in virtual reality rehabilitation environments," in *proceedings of the 10th international conference on virtual reality continuum and its applications in industry*, 2011, pp. 343–346.

[18] Y. Zhang and S.-B. Tsai, "Application of adaptive virtual reality with ai-enabled techniques in modern sports training," *Mobile Information Systems*, vol. 2021, 2021.

[19] B. Ferreira and P. Menezes, "An adaptive virtual reality-based serious game for therapeutic rehabilitation," 2020.

[20] F. Loch, M. Fahimipirehgalin, J. N. Czerniak, A. Mertens, V. Villani, L. Sabattini, C. Fantuzzi, and B. Vogel-Heuser, "An adaptive virtual training system based on universal design," *IFAC-PapersOnLine*, vol. 51, no. 34, pp. 335–340, 2019.

[21] H. Shum and S. B. Kang, "A review of image-based rendering techniques," vol. 4067, 05 2000, pp. 2–13.

Appendix A
A simple example scenario as JSON

```
{
  "entities": [
    {
      "id": 1,
      "class": "Fire",
      "name": "Fire 1",
      "position": "(0, 0, 0)",
      "rotationOnPlane": "0",
      "properties": [
        { "id": 1, "key": "intensity",
          "value": "10" }
      ]
    },
    {
      "id": 2,
      "class": "Person",
      "name": "Person 1",
      "position": "(10, 0, 10)",
      "rotationOnPlane": "30",
      "properties": [
        { "id": 2, "key": "panic", "value":
          "False" }
      ]
    }
  ],
  "reactions": [
    {
      "id": 1,
      "condition": {  "sourcePropertyId":
        "1", "value": "20", "operator":
        ">=" },
      "action": { "targetPropertyId": "2",
        "newValue": "true" },
    },
  ]
}
```