VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes

Xiaoyin Wang
xiaoyin.wang@utsa.edu
University of Texas at San Antonio
San Antonio, Texas, USA

ABSTRACT

Virtual Reality (VR) is an emerging technique that attracts interest from various application domains such as training, education, remote communication, gaming, and navigation. Despite the ever growing number of VR software projects, the quality assurance techniques for VR software has not been well studied. Therefore, the validation of VR software largely rely on pure manual testing. In this paper, we present a novel testing framework called VRTest to automate the testing of scenes in VR software. In particular, VRTest extracts information from a VR scene and controls the user camera to explore the scene and interact with the virtual objects with certain testing strategies. VRTest currently supports two built-in testing strategies: VRMonkey and VRGreed, which use pure random exploration and greedy algorithm to explore interact-able objects in VR scenes. The video of our tool is available on Youtube at https://www.youtube.com/watch?v=TARqTEaa7_Q

KEYWORDS

Software Testing, Virtual Reality, Scene Exploration

ACM Reference Format:
Xiaoyin Wang. 2022. VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510454.3516870

1 INTRODUCTION

Virtual Reality (VR) is an emerging technique [10] which has many different application scenarios, such as gaming, virtual exhibition and tour, training, education, product design, and remote communication. A recent market report [3] estimates that the total value of VR market has reached 11.52 billion dollars in 2019, including 1.9 billion dollars of VR software market [5]. Furthermore, the market value is expected to grow at a high growth rate of 48.7% per year in the following five years. The pandemic of COVID-19 virus further accelerated the adoption of VR techniques. In 2020, Thousands of apps are uploaded to Google Play [2], Apple Store [1], and Oculus Market [4]. These apps have been downloaded by more than 171 million users from all over the world [9]. Like all other types of software, virtual reality software also needs testing to validate and enhance its quality. However, unlike in many other software domains (e.g., software libraries, server software, GUI software) where many automatic and semi-automatic testing techniques have been developed and partially adopted, automatic testing techniques for VR software are not well studied.

VR software typically consists one or several VR scenes. Each VR scene represents a three dimensional space in which virtual objects (comparable to icons/controls in GUI software [22]) are placed and moved. Users can operate the software by interacting with the virtual objects typically through pointer clicking (i.e., keeping a white point at the center of the camera view pointing to an interact-able object for a short amount of time).

In this paper, we present an automatic testing framework VRTest to test VR scenes in VR software, with two built-in testing strategies VRMonkey and VRGreed. In particular, VRTest automatically and periodically (to handle object movement) locates all virtual objects in the virtual space, and move / rotate the camera according to the testing strategy. It should be noted that VRTest needs to track both interact-able objects and non-interact-able objects because the latter may block the route of camera movement and occlude interact-able objects. Based on VRTest, VRMonkey is a pure random exploration testing strategy which mirrors Monkey for Android platform and we use it as a baseline technique. VRGreed uses a greedy algorithm to have the camera visit and trigger all interact-able virtual objects, starting from the closest object.

We evaluated VRTest with two testing strategies on five top VR software projects (based on scores) from UnityList, which is a large repository of Unity-based VR/AR open source software projects. Although our evaluation focuses on Unity-based software, we believe this scope of subject selection is reasonable because Unity dominates VR software development with over 60% market share according to multiple sources [6, 7]. The evaluation results show that VRTest framework is flexible enough to incorporate different testing strategies. Furthermore, VRGreed enhanced interact-able object coverage by 55 percentage points over VRMonkey.

To sum up, this paper makes the following contributions.

- We explore and summarize the major challenges in automatically testing VR scenes.
- We develop a framework to automatically testing VR scenes which can be incorporated with different testing strategies.
- We develop two built-in testing strategies based on purely random exploration (VRMonkey) and greedy exploration (VRGreed).
- We prepare a dataset with five top VR software projects that can be reused in future research in this area.

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05…$15.00
https://doi.org/10.1145/3510454.3516870

- We perform an initial evaluation of VRTest on five top VR software projects.

2 FRAMEWORK

In this section, we describe our VRTest framework which automatically explores a virtual scene in a VR software. VRTest provides a set of basic operations to control and configure the testing process. The overview of VRTest framework is presented in Figure 1. From the figure, we can see that the VRTest Framework consists of five major components: the VR Scene Monitor to fetch information (i.e., run-time state) of the VR scene, the Virtual Object Instrumenter which instruments interact-able virtual objects with state reporters, the Exploration Controller which controls all the information transmission and the pace of the exploration, the Test Configuration Interface, and the Testing Technique Interface which provides scene information feedback and three basic actions (move, rotate, and trigger) for the testing technique to implement.

2.1 VR Scene Monitor

The *VR Scene Monitor* component extracts information about virtual objects in the VR scene. Since virtual objects can be created and destroyed at run time, and moving from one location to another, the information extraction process is periodic. In particular, this component extracts the following major types of information because they can be useful for various testing techniques.

- **Bounding Boxes of Virtual Objects.** The monitor extracts the bounding boxes (the smallest rectangular cuboid enclosing the object) of all virtual objects that have a renderer (so that it is visible and will block eye sight). The monitor further separates virtual objects with colliders from others because these objects may block the user (and the camera) from moving through it. It should be noted that the bounding box (See Figure 2) is just an approximation of the virtual object's shape to simplify following computation. Also, we do not handle objects with transparent renderers separately although they may not block sight. We believe these are proper approximations due to the large variety in virtual objects' shape and transparency, and because such approximations do not cause false positives / negatives in testing but may just slow down the exploration process (e.g., when an object is considered blocked by another object but it is actually visible with current camera location and angle).
- **Position of Virtual Objects.** The monitor extracts the three dimensional coordinates of all virtual objects with renderers. It provides both an object's registered coordinates (its position attribute), and the center coordinates of its bounding box. For virtual objects with irregular shapes, the test framework may need to try both positions to reach them.
- **Interact-able Properties of Virtual Objects**: The monitor further identifies the interact-able virtual objects from the others. These interact-able objects are provided to the testing techniques as their exploration targets.
- **Type Information of Interact-able Virtual Objects**: In Unity-based VR software, many virtual objects can be programmatically created from a single template called prefabs. A virtual object can also be created by cloning another existing object. Since these object copies typically have the same behavior and the same set scripts attached to them, triggering events on them may cover exact the same object behavior and code portions. So it can be inefficient to trigger events on them multiple times, and the monitor provides such information to the testing strategies for them to do optimizations.

2.2 VR Object Instrumenter

There are a lot of uncertainties when VRTest tries to trigger a pointer click event on a virtual object. For example, the object may be of irregular shape so targeting the user camera at its position may not trigger the event. Also, some renderers may be transparent so the user camera may accidentally triggered some other events. To make sure VRTest always have precise and updated information about which virtual objects have already been covered, we design an instrumenter component in VRTest. In particular, the instrumenter will instrument all interact-able virtual objects by adding a state-change reporter (an additional reporting event handler method) for each of their registered events. The state-change reporter will report to VRTest once the corresponding event is triggered so that VRTest always knows which objects and events have been already triggered. The code we use to insert the state-change reporter is shown as follows.

```
...
EventTrigger r = go.GetComponent<EventTrigger>();
EventTrigger.Entry entry = new EventTrigger.Entry();
entry.callback.AddListener((eventData) => { UpdateTrigger();});
r.triggers.Add(entry);
...
```

Listing 1: The Code to Insert State-Change Reporter

In particular, we first fetch the `EventTrigger` component `r` from an interact-able virtual object `go`, and then insert a pointer to our reporting event handler function (i.e., `UpdateTrigger`) into the list of triggers in `r`.

2.3 Exploration Controller

The exploration controller is the core component of VRTest and it controls the whole testing process. It receives updated information from VR Scene Monitor and State-Change Reporters, and provides the information to the testing strategies it incorporates. It should be noted that the Unity framework periodically calls `Update` (every frame) and `FixedUpdate` (every 0.02 second) methods to refresh the VR scene. VRTest embeds its exploration controller into the `FixedUpdate` method so that exploration process is synchronized with the frame refresh of the VR scene. We choose `FixedUpdate` instead of `Update` because the former has a fixed invocation gap not affected by run-time fps (frame per second) and thus it provides better control of physical movement.

If the action has been finished, VRTest will first extract VR scene information from the monitor, call the instrumenter to instrument any newly found virtual objects that has not been instrumented, and then ask the testing technique to decide what actions to be performed next (i.e., trigger, rotate, move, or any combination of

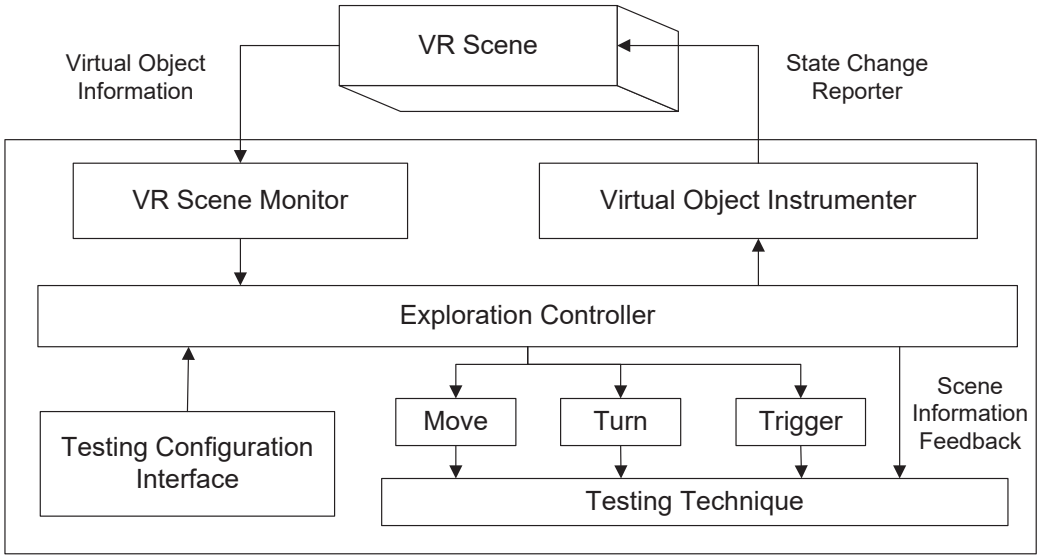

Figure 1: The Overview of VRTest Framework

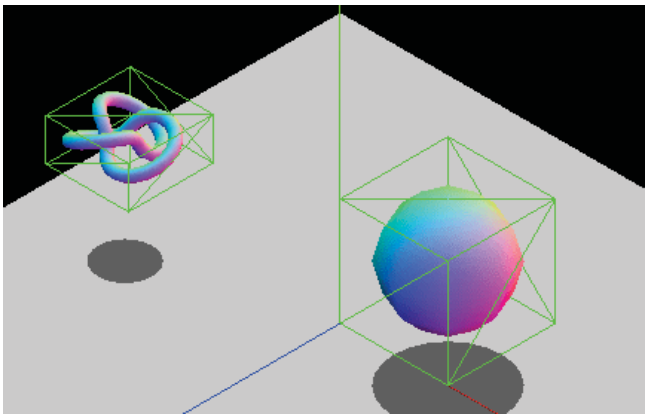

Figure 2: Bounding Boxes of Virtual Objects from developer.mozilla.org

them). After the action is determined, the controller will perform the action step by step in the following cycles of `FixedUpdate`.

2.4 Testing Configuration

VRTest allows developers to configure VRTest based on the requirement and characteristics of the VR scene to be tested. Some major configuration items are listed as follows.

- **Moving Scope.** The moving scope defines the range of positions the user camera can move to, and it needs to be taken into account when a testing strategy calculates feasible routes between two positions. The scope can be very different according to the feature of the VR software. For example, a flying simulation software application may allow the user camera to move in all three dimensions to a large extent, but a driving / walking simulation software application may allow only movement on the X and Y axes, but does not allow any Z-Axis movement. Some other software may have more fixed moving pattern (e.g., along a line or route) or even do not allow any movement (e.g., some shooting games). The VRTest framework allows a developer to manually specify the moving scope.
- **Rotation Scope.** Similar to moving scope that limits the user camera's position, the rotation scope limits the user camera's watching angle. This is more consistent among different VR applications. Typically, Y-axis rotation (turning head to the left and right) can go up to 180 degree on both sides, and X-axis rotation (turning head up and down) can go up to 90 degrees. The reason is that this rotation scope is roughly the same with that of a human being's head. So in our framework we also set the rotation scope accordingly by default. Z-axis rotation (tilting head to the left or right) is often allowed to a small degree, but since it does not affect the testing process (tilting your head will not allow you to see more than not tilting), we simply do not allow Z-axis rotation for simplicity. By default, we set the rotation scope with X-axis rotation between -90 degree and 90 degree, Y-axis rotation between -180 degree and 180 degree, and no rotation for Z-axis.
- **Moving / Rotation Speed.** Moving and rotation speed limits the speed of moving and rotating the user camera. By default, we use 1 meter per second (1 unit in Unity-based VR scene represents 1 meter) as the moving speed because it is the normal walking speed. We use 10 degree per second as the rotation speed as it is the maximal rotation speed to avoid motion sickness [12]. It should be noted that, if there is no need to video record the testing process for further analysis, a developer can increase such speeds to reduce testing time.

Table 1: Basic Information of Evaluation Subjects

Name	#Source Files	LOC	#Virtual Objects
UnityVR	129	25.6K	36
UnityVREscapeRoom	207	31.2K	109
unity-vr-maze	7	503	26
VRND_Night_at_the_Museum	173	30.5K	32
unity-vr-cave-puzzler	7	8.0K	27

However, this configuration value does not affect the testing efficiency comparison among testing strategies because it will accelerate or slow down all strategies with a same rate. This mainly a balance between testing efficiency (the higher the speed, the higher the efficiency) and user experience.

2.5 Testing Strategy Interface

VRTest provides an interface for any testing technique to implement. The interface includes three methods: `Rotate`, `Move`, and `Trigger`. Different testing techniques can implement these three methods to explore the VR scene with different strategies. VRTest provides a default `Trigger` implementation for pointer click events, but it can be easily extended with new types of events and different ways to trigger the events. Furthermore, VRTest provide interfaces for the testing technique to acquire information about the VR scene (i.e., all the information collected from VR Scene Monitor, State-Change Reporter, and Testing Configuration Interface), in case it is needed when implementing the testing strategy.

3 EVALUATION

To evaluate our framework and testing strategies, we apply VRTest with two testing strategies on five top VR software projects from UnityList[1].

We collected our subject projects from UnityList [8] because it is the largest repository of open source VR software projects. From UnityList, we followed the ranking of featured scores, and considered only projects with at least one virtual object with at least one event triggers. The basic information of five subject projects in our evaluation is presented in Table 1. In the able, we present the number of source files, the number of lines of code, and the number of static virtual objects / prefabs (dynamic virtual objects are typically created by cloning static virtual objects / prefabs), respectively. To perform the evaluation, we use Unity version 2019.4.2f1 with Visual Studio 2017 (for compilation of C# source code) and run the experiment on a computer with Intel Core i7-6500U CPU, 8GB of memory, and Intel HD 520 Graphics card. We set a timeout of 100 seconds.

We measure the effectiveness of testing by the interact-able object coverage (i.e., the proportion of triggered interact-able objects among all interact-able objects). For interact-able object coverage, we count objects of the same type as one. The results are shown in Table 2. In the table, Column 2 and 3 present the coverage of VRMonkey and VRGreed, respectively. From the table we can see that

[1]The implementation of VRTest and the evaluation dataset can be downloaded from https://sites.google.com/view/vrtest2021

Table 2: Interact-able Object Coverage of VRMonkey and VRGreed

Name	VRMonkey	VRGreed
UnityVR	25%	75%
UnityVREscapeRoom	22%	55%
unity-vr-maze	0%	55%
VRND_Night_at_the_Museum	0%	100%
unity-vr-cave-puzzler	0%	100%

both VRMonkey and VRGreed can cover interact-able objects at run time, but VRGreed has much higher coverage than VRMonkey, which is as expected.

4 DISCUSSION

Our testing framework and testing techniques currently focus on the pointer click event type as it is the most commonly supported event type. There are some other event types supported by certain devices, such as the grabbing event which allows a user to grab certain virtual object with their virtual hand, and the colliding event that allows a user to push or collect certain virtual objects when the user camera is at the same position or close to an existing virtual object. Since these events are mainly contact-based events (i.e., the user camera needs to be very close to the virtual object to trigger the event), they are less complicated to trigger compared with pointer clicks as we do not need to consider scenarios such as object occluding. A more complicated case is when the virtual objects must be interacted in certain order to lead to an outcome. None of our three testing techniques intentionally handle such interaction orders, so whether the outcome can be triggered may largely rely on repetitive triggering of events on interact-able virtual objects when the methods associated with them are still not covered. In the future, we plan to use static analysis to identify the dependencies between event handlers. Based on the dependencies, VRTest would be able to trigger events in more proper order to expose more software behaviors.

5 RELATED WORKS

We are not aware of existing efforts in the area of VR software testing, but there are some research on game testing. Wuji [25] is a framework to automatically test games based on evolutionary algorithms and reinforcement learning. It explores the game spaces and branches as well as making progress by passing stages. Testmig [19] migrate test cases across mobile devices but their approach cannot be extended directly to VR devices. Zhao et al. [24] proposed an approach to enhancing playing tactics in game testing by learning from player action sequences. Bergdahl et al. [11] proposed an approach to augment existing manually written test scripts with reinforcement learning. Molina et al. [15] proposed VRDepend, an automatic approach to extract dependency among virtual objects, which may used for analysis and testing of VR software. However, all of the above approaches mainly focus on game tactics and are designed for 2D games, so when applied to 3D software they still face the challenge of flexible camera movement/rotation and accessing out-of-view and occluded objects, which are the focus of this paper.

There also have been some empirical studies on VR software and video game software. Murphy-Hill et al. [16] performed a study on video game developers to understand the challenges in video game development and how they are different from traditional software development. Washburn et al. [21] studied failed game projects to find out the major pitfalls in game development. Lin et al. [14] studied the common updates in steam platform to understand the priority of game updates. Rodriguez and Wang. [20] performed an empirical study on open source virtual reality software projects to understand their popularity and common structures. Pascarella et al. [18] studied open source video game projects to understand their characteristics and the difference between game and non-game development. Zhang et al. [23] studied possible solutions to detect potential privacy leaks in mobile augmented reality apps. Li et al. [13] studied the characteristics of bugs in web-based extended reality apps. Nusrat et al. [17] studied the performance optimizations from version history of real-world VR projects and summarized major performance issues that new VR developers should avoid.

6 CONCLUSION

VR software is gaining more and more usage scenarios, but its quality assurance techniques still fall behind. In this paper, we propose a novel framework called VRTest to automatically test VR software. VRTest extracts information from the VR scene and controls the user camera to explore the scene and interact with the virtual objects. Based on VRTest, we further developed two testing strategies: VRMonkey and VRGreed, which use pure random strategy and greedy algorithm, respectively. We also performed an initial evaluation of the testing strategies on five VR software projects from UnityList. New testing strategies can be easily added by extending VRTest with new implementations of `Move`, `Turn`, and `Trigger` functions.

In the future, we plan to work on the following directions. First of all, for the VRTest framework, we plan to extend it to support more types of events such as grabbing events and colliding events. Second, we plan to extend testing strategies to consider more global information in the VR scene, and we plan to further enhance our strategies by using AI-based or search-based techniques which have been shown effective in GUI testing to acquire a globally optimized route. Third, we plan to evaluate our framework with more subjects and software projects that are not based on Unity. Fourth, certain software behavior may be exposed only when events are triggered in certain order, so we plan to use static analysis to identify the dependencies between event handlers. Based on the dependencies, VRTest would be able to trigger events in a certain order to expose more software behaviors.

ACKNOWLEDGEMENTS

This research work is supported in part by NSF Grant CCF-2007718.

REFERENCES

[1] 2020. Apple App Store. https://www.apple.com/ios/app-store/. Accessed: 2020-06-30.
[2] 2020. Google Play. https://play.google.com/store. Accessed: 2020-06-30.
[3] 2020. Mordor Intelligence Report on Virtual Reality Market. https://www.mordorintelligence.com/industry-reports/virtual-reality-market. Accessed: 2020-06-30.
[4] 2020. Oculus App Store. https://www.oculus.com/experiences/quest/. Accessed: 2020-06-30.
[5] 2020. Statistica Report on Virtual Reality Software Market. https://www.statista.com/statistics/550474/virtual-reality-software-market-size-worldwide/. Accessed: 2020-06-30.
[6] 2020. Unity Engine: A Unicorn Powering the Video Game and VR/AR Economy. https://digital.hbs.edu/platform-digit/submission/unity-engine-a-unicorn-powering-the-video-game-and-vr-ar-economy/. Accessed: 2020-12-30.
[7] 2020. Unity IPO aims to fuel growth across gaming and beyond. https://techcrunch.com/2020/09/10/how-unity-built-a-gaming-engine-for-the-future/. Accessed: 2020-12-30.
[8] 2020. UnityList. https://unitylist.com/. Accessed: 2020-06-30.
[9] 2020. VR user statistics. https://techjury.net/blog/virtual-reality-statistics/#gref. Accessed: 2020-06-30.
[10] Leif P Berg and Judy M Vance. 2017. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual reality* 21, 1 (2017), 1–17.
[11] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. 2020. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*. IEEE, 600–603.
[12] Eunhee Chang, Hyun Taek Kim, and Byounghyun Yoo. 2020. Virtual reality sickness: a review of causes and measurements. *International Journal of Human–Computer Interaction* 36, 17 (2020), 1658–1682.
[13] Shuqing Li, Yechang Wu, Yi Liu, Dinghua Wang, Ming Wen, Yida Tao, Yulei Sui, and Yepang Liu. 2020. An exploratory study of bugs in extended reality applications on the web. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 172–183.
[14] Dayi Lin, Cor-Paul Bezemer, and Ahmed E Hassan. 2017. Studying the urgent updates of popular games on the Steam platform. *Empirical Software Engineering* 22, 4 (2017), 2095–2126.
[15] Jacinto Molina, Xue Qin, and Xiaoyin Wang. 2021. Automatic Extraction of Code Dependency in Virtual Reality Software. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 381–385.
[16] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. 2014. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?. In *Proceedings of the 36th International Conference on Software Engineering*. 1–11.
[17] Fariha Nusrat, Foyzul Hassan, Hao Zhong, and Xiaoyin Wang. 2021. How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 473–485.
[18] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. 2018. How is video game development different from software development in open source?. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 392–402.
[19] Xue Qin, Hao Zhong, and Xiaoyin Wang. 2019. Testmig: Migrating gui test cases from ios to android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 284–295.
[20] Irving Rodriguez and Xiaoyin Wang. 2017. An empirical study of open source virtual reality software projects. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 474–475.
[21] Michael Washburn, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2016. What Went Right and What Went Wrong: An Analysis of 155 Postmortems from Game Development. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. 280–289.
[22] Xusheng Xiao, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. 2019. Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 257–268.
[23] Xueling Zhang, Rocky Slavin, Xiaoyin Wang, and Jianwei Niu. 2019. Privacy Assurance for Android Augmented Reality Apps. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 114–1141.
[24] Yan Zhao, Weihao Zhang, Enyi Tang, Haipeng Cai, Xi Guo, and Na Meng. 2021. A Lightweight Approach of Human-Like Playtesting. *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2021).
[25] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM Conference on Automated Software Engineering (ASE)*. 772–784.