MRAT: The Mixed Reality Analytics Toolkit

Michael Nebeling[1a,b]**, Maximilian Speicher**[1a,2]**, Xizi Wang**[1a]**, Shwetha Rajaram**[1a]**, Brian D. Hall**[1a]**,
Zijian Xie**[1c]**, Alexander R. E. Raistrick**[1b]**, Michelle Aebersold**[1a,d]**, Edward G. Happ**[1a]**,
Jiayin Wang**[1a]**, Yanan Sun**[1a]**, Lotus Zhang**[1a]**, Leah E. Ramsier**[1a]**, Rhea Kulkarni**[1a]

[1] University of Michigan   [2] C&A Europe
[a] School of Information   [b] College of Engineering
[c] College of Literature, Science, and the Arts   [d] School of Nursing
nebeling@umich.edu — https://mi2lab.com

ABSTRACT

Significant tool support exists for the development of mixed reality (MR) applications; however, there is a lack of tools for analyzing MR experiences. We elicit requirements for future tools through interviews with 8 university research, instructional, and media teams using AR/VR in a variety of domains. While we find a common need for capturing how users perform tasks in MR, the primary differences were in terms of heuristics and metrics relevant to each project. Particularly in the early project stages, teams were uncertain about what data *should*, and even *could*, be collected with MR technologies. We designed the *Mixed Reality Analytics Toolkit* (MRAT) to instrument MR apps via visual editors without programming and enable rapid data collection and filtering for visualizations of MR user sessions. With MRAT, we contribute flexible interaction tracking and task definition concepts, an extensible set of heuristic techniques and metrics to measure task success, and visual inspection tools with in-situ visualizations in MR. Focusing on a multi-user, cross-device MR crisis simulation and triage training app as a case study, we then show the benefits of using MRAT, not only for user testing of MR apps, but also performance tuning throughout the design process.

Author Keywords

Augmented/virtual reality; interaction tracking; user testing.

INTRODUCTION

Given the increased availability of new AR/VR headsets and smartphone-based platforms, there is a renewed interest in research on mixed reality (MR) and its applications [24, 36]. Most studies to date are research experiments taking place in heavily instrumented lab environments [6, 11, 13, 38]. A 10-year survey by Dey *et al.* [11] reviewed 369 AR user studies and found that 75% were lab-based evaluations. Although the value of conducting user studies in more realistic environments has long been recognized [7, 13, 40], there are still many barriers that need to be overcome.

First, while there is a wide variety of tools for AR/VR interface design and development [27], there is still a lack of tool support for evaluation. In the 10 years of AR user studies reviewed by Dey *et al.* [11], there were few common evaluation tools, and most studies required the researchers to use video analysis or build ad-hoc solutions on top of their research prototypes to gather, visualize, and analyze quantitative data. New commercial tools produce analytics of MR apps [10, 3, 42], but only few operate at the interaction level required for user studies and usability evaluation [12, 14], and the few tools that do, almost exclusively produce gaze and click heatmaps.

Second, the comprehensive survey by Dey *et al.* found almost no heuristic evaluations, suggesting there is a need for heuristics and metrics to help assess the MR user experience. While research on web [5, 8, 25], mobile [29, 28], tabletop [16, 19], and Kinect-based interfaces [20, 26] has contributed to building a better understanding of how to measure and improve the user experience [30], what constitutes a good MR experience and how to measure it is still an open research question.

This paper makes three primary contributions: First, we conducted interviews with eight university research, instructional, and media teams working on MR applications in many different domains: film, visualization, crisis informatics, dentistry, landscaping, architecture, media, and nursing. These interviews helped us establish requirements for the evaluation of MR interfaces in terms of *contexts* (ranging from classroom, to clinical, to outdoor settings), *tasks* (ranging from formal, experimental designs to informal, exploratory evaluations), and *metrics* (ranging from performance-based metrics such as task completion times, to location-based metrics such as common navigation paths, to interaction-based metrics such as gestures and speech commands), informing the design of heuristics and more comprehensive MR evaluation tools.

Second, we designed the *Mixed Reality Analytics Toolkit* (MRAT) to assist the entire process starting with the instrumentation of an MR app under evaluation, to the definitions of tasks and metrics for data collection, to visualizations for analysis. MRAT is targeted at researchers and designers who wish to evaluate MR apps. We designed MRAT with non-technical users in mind, implementing it on top of Unity which is popular for MR app development, but exposing MRAT's core functionality in the Unity Editor, where most aspects can be configured visually without the need for programming. Given

our interviewees' diverse requirements, we aimed for flexible solutions to allow MRAT to be configured for a variety of evaluation goals. We introduce our enabling concepts and techniques based on *task markers* to guide users through studies and automatically segment data by task, *tracking scopes* to optimize data collection by limiting the number and types of objects and interactions to be tracked by MRAT, and *inspection tools* for live observation and post-hoc analysis.

Third, we present our MRAT-CRISIS case study detailing how we used MRAT to instrument an MR crisis simulation and triage training app created in collaboration with two of the interviewed teams. We chose this app because its complexity allowed us to demonstrate and test many of MRAT's features. Specifically, we explored how MRAT could best be configured to produce relevant metrics and visualizations to allow instructors to observe student teams remotely as students were collaboratively solving a crisis scenario, and to understand how teams differed in terms of behaviors and performance.

RELATED WORK
Since Milgram's original review of MR applications with characterizations along the reality–virtuality continuum [24], there have been many surveys on how MR applications are designed and evaluated [11, 12, 36, 37, 43]. Surveys of AR usability studies by Duenser and Billinghurst [12] and Dey et al. [11] found similarities in study designs and evaluation methods, but little commonality in terms of tools and heuristics used between studies. Voit et al. [43] identified challenges and opportunities unique to MR evaluations when compared to traditional online, lab, and field experiments. Finally, Speicher et al.'s analysis of 68 MR papers resulted in a new conceptual framework for assessing MR experiences [36].

Just sampling 10 AR/VR papers from CHI 2019, we find an overwhelming use of qualitative assessment tools. Common ones include NASA-TLX and various presence and immersion questionnaires to assess user experience and embodiment in VR [2, 34]. Video analysis is also common, e.g., used to analyze gesture performance [39] or understand participants' workflows in designing for VR [15, 22]. However, there are only a few common metrics and even fewer specific to MR. Task completion time and task error rates were still common, e.g., in controlled experiments [39] or to study learning effects for different tasks [33]. Three studies implemented user tracking tools for collecting participants' VR motion data [21, 33, 35], then each developed a unique approach to data analysis.

We also performed a directed literature search related to our MRAT-CRISIS case study presented later. Shaw *et al.* [35] developed a VR fire evacuation simulation, collecting positional data, video/audio recordings, and questionnaire data to evaluate user performance. Tayama *et al.* [41] developed a mobile triage training simulation and collected questionnaire data to evaluate system usability and pre- and post-tests to assess learning. Though these two studies had similar data collection needs, they each implemented ad-hoc solutions on top of the systems they wanted to evaluate.

Prior work has developed interaction tracking techniques and tools for different platforms and modalities. For web interfaces, Atterer *et al.* [5] developed principles that were later used to determine users' goals in web searches [18], compare user interactions to predefined optimal logs [8], and automatically adapt interfaces from desktop to mobile configurations [28]. These interaction tracking principles were then extended to new modalities enabled by sensors like Kinect. For example, ChronoViz [17] and Kinect Analysis [26] provide means to track and visualize Kinect interactions in timeline and heatmap visualizations. Similar techniques were used in MAGIC [4] and GestureAnalyzer [20] but with a focus on visual analytics for gesture elicitation. While these existing tools do not specifically target MR apps, they informed key components of MRAT for interaction tracking and session inspection.

REQUIREMENTS ELICITATION
To inform the design of MRAT, we conducted interviews with teams composed of research, instructional, and media teams working on MR projects in different domains across our university. We wanted to understand common requirements and were hoping to make MRAT applicable to many use cases.

Method & Participants
We recruited 8 research, instructional, and media teams working in various disciplines across our university. We contacted participants through two AR/VR interest groups' mailing lists, one for a community of practice and the other focused on research and instruction. Domain areas represented by these teams were film, visualization, crisis informatics, dentistry, landscaping, architecture, media, and nursing. We set up separate 1–2 hour interviews with these teams. Each team was asked to select one AR or VR project (henceforth referred to as *"MR app"*) to focus the interviews. Our interviews were structured into four blocks: project goals, target users, insights each team hoped to gain, and methods used for evaluation.

Findings
The first interesting finding was the diversity of project goals and metrics among just these 8 teams. We assigned aliases to capture the topic of each project. Team FILM STUDY recreated a classic film scene in VR for a filmmaking course. Team PAIN INDICATION used an AR brain model to visualize indications of pain in dental patients' brains in a research project. Team CRISIS SIMULATION wanted to use AR to create a realistic disaster in the classroom and evaluate how students solve a crisis scenario. Team LANDSCAPE MODELING had conducted controlled experiments on the perceived biodiversity of modeled landscapes and wanted to test an app they were creating for designing landscapes in VR. Team OFFICE SPACE used VR to test subjective perceptions of different office environments in a research project. Team TRIAGE TRAINING worked on a triage training application simulating facial features of a stroke using AR markers on a manikin. Team CAMPUS TOUR developed and deployed a mobile app with several MR features such as AR overlays for landmarks and 360 videos about the university's history. Team 3D LAB supported research teams across the university in the implementation of various MR apps, including some of the above.

(R1) Need for evaluations beyond usability testing. With the exception of one team (CAMPUS TOUR), it became clear

that standard usability measures (efficiency, effectiveness, and satisfaction) were not the main concern. The teams typically had larger research questions, such as whether a user's experience changed between AR and VR, or how to promote and assess problem solving strategies and creativity.

(R2) Need for quantitative and spatial data collection. All teams mentioned a variety of quantitative data they wished to collect on app usage (e.g., where users walked, what users looked at and clicked). Most collected qualitative data in the form of user experience reports, interviews, and surveys. They all saw value in collecting information within the MR app about interactions that are difficult to capture through video.

(R3) Need for tracking diverse interactions. Team FILM STUDY wanted students to be able to use the VR headset as a 6-DOF camera for retaking film scenes. In LANDSCAPE MODELING and OFFICE SPACE, users' gaze was of importance to understand how people perceived certain aspects of architecture. Team CRISIS SIMULATION was interested in multi-user verbal and social interactions, asking whether time users spent in proximity to each other could be calculated. Studying users' perceptions of a 3D landscape on a large screen compared to VR and later AR, Team LANDSCAPE MODELING wanted to track interactions across different MR platforms and devices.

(R4) Need for flexible task definitions. As also pointed out by Team 3D LAB, studies involved many different tasks and used study designs ranging from controlled experiments to open-ended exploratory studies. Team LANDSCAPE MODELING presented a controlled experiment with very specific tasks in different conditions. Team CRISIS SIMULATION used timed injects of crisis events according to a protocol developed for the simulation in class. Team TRIAGE TRAINING mentioned a number of time limits for clinicians to successfully pass the simulation of a stroke. Team FILM STUDY was expecting students to experiment with multiple different takes and be creative, but there were no specific tasks at this stage.

(R5) Need for key performance indicators (KPI). Researchers mentioned many types of data specific to their projects but relatively few concrete metrics. Time spent looking in a certain direction or standing at a certain location was a metric mentioned by almost all teams. Four teams also wanted to analyze trends and patterns between groups of users, such as frequent locations or targets. Teams wished for MRAT to guide them in the development of heuristics and metrics.

MRAT: THE MIXED REALITY ANALYTICS TOOLKIT

This section presents MRAT, first in terms of the main processes it enables, then in terms of the system components (Fig. 1). At the highest level, MRAT supports three processes: *Interaction Tracking*, *Task Definition*, and *Session Inspection*. Each process can be configured via visual editors in Unity and scripts provided by MRAT (labeled in blue in Fig. 1).

At the system level, MRAT consists of a *Unity Package* that can easily be added to an existing MR app in Unity, and a *Server* that stores user sessions and opens them in a web-based *MRAT Dashboard*. MRAT's tight integration with the MR app under evaluation allows session inspection to be done in real-time. The MRAT Dashboard shows selected tracking data in
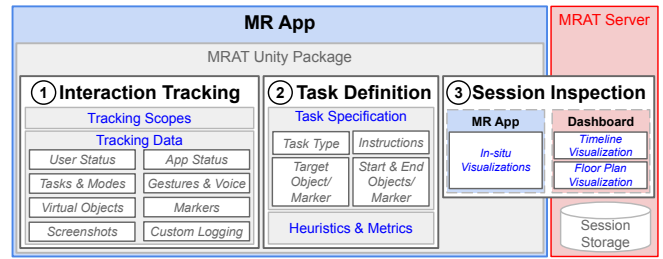

Figure 1. MRAT's three processes and enabling system components

Timeline and *Floor Plan* visualizations side by side. MRAT's *In-situ Visualizations* superimpose events from tracking data in MR at the precise location where they were captured. Below, we detail each process and give examples from the interviewed teams to explain our design rationale.

1. Interaction Tracking ⇒ *Requirements R1, R2, R3*
The first main process enabled by MRAT is interaction tracking. Given the diverse requirements from the interviews, we developed a comprehensive set of system components able to handle the seamless collection of a variety of interaction data.

Tracking Scopes
Many interactions in MR happen *implicitly* [23], based on user movement, gaze, time, or system events, as opposed to *explicit* interactions such as a user's gestures or voice commands. In complex MR apps like games or multi-user experiences, there are typically tens, if not hundreds, of virtual objects present, and users may trigger large amounts of events. A key challenge when engineering MRAT was to enable data collection without impacting system performance, and thus user experience, of the MR app under evaluation. For HoloLens apps, Microsoft mandates a consistent frame rate of 60 frames per second (FPS) [9]; a qualified Oculus VR app must maintain 45 FPS to meet minimum specifications [32].

This tight performance envelope is attained in MRAT based on a notion of tracking scopes. Globally, we distinguish *per frame*, *per event*, and *per interval* tracking scopes, limiting the collection for every frame only to when strictly necessary, using event-based triggering when available, and otherwise using a configurable time interval. MRAT can also toggle tracking features *per task* (e.g., to listen for gesture events or voice commands only for a particular task, while they would go unnoticed in another task). Objects can be tracked on a *per instance* or *per type* basis. The *per instance* scope tracks only a certain instance of an object (e.g., a specific injury marker of one player in the TRIAGE TRAINING project), as opposed to tracking all instances of a specific type of object (e.g., all virtual objects or marker-tagged physical objects for collecting supplies in the CRISIS SIMULATION project).

Tracking Data
Based on the above tracking scopes, MRAT collects the following classes of interaction data:

User & App Status. MRAT continuously tracks the user's position, rotation, and gaze in the form of `StatusUpdate` events per a configurable time interval (2 seconds per default). These status events support two common KPIs demanded by

the teams: time spent looking in a certain direction or standing at a certain location. MRAT also collects app status data in terms of frame rate and memory use for every frame. It can be configured to fire `PerformanceWarning` events if thresholds are not met (30 FPS per default). We use a lower than recommended FPS threshold per default to limit performance warnings to a critical state. This can be helpful for the early design stages when experimenting with new features in an MR app. Marker tracking, large numbers of objects, high-poly 3D models, and particle systems are all demanding on system performance. As described later, MRAT's performance warnings can help tune data collection to not interfere with the execution and rendering of the MR app's logic and content.

Tasks & Modes. To enable MRAT's task definition and session inspection components, MRAT collects a variety of task-related data. This includes when a task is *started*, *skipped*, or *completed*, which can be detected automatically based on scripts provided by MRAT (e.g., for detecting voice commands or markers). When a task is completed, the tracking component reports information about duration, accuracy, and whether the task was successful or not, as determined via the task definition components. Further, MR apps can report modes to MRAT, which then automatically labels all data being collected with the currently active mode. Modes can span across tasks and can be mapped to interface modes of the MR app under evaluation (mode-error slips are a common usability issue [31]), or can be flexibly used to represent conditions or phases of a study at a higher-than-task level (in our case study around a multi-user, cross-device MR app reported later, we map modes to stages in a shared simulation).

Gestures & Voice Commands. MRAT automatically listens for a variety of standard gesture events, such as HoloLens clicker input and air taps or VR motion controller events. This supports another common KPI demanded by the teams: which virtual objects are commonly targeted. If a virtual object is focused via the user's gaze at the same time, a tap is recorded as a *click*, otherwise as a *tap without target*. To detect potentially *missed gestures* on HoloLens, MRAT tracks the state of the user's hand, detecting when it moves into the FOV with the index finger pointing up, but does not perform a gesture. Additionally, the tracker captures *voice commands* and *focus* events, such as those generated by an MR app's gaze cursor or VR controller's laser.

Virtual Objects. For a comprehensive understanding of what happens in an MR app, MRAT can be configured to track virtual objects in a scene. This includes when objects are created (which adds them to the "tracking list") when an object's position or orientation changes, and when an object is removed from the scene. These data are sent as `ObjectUpdate` events in a configurable interval, only if there has been a change.

Object & Task Markers. MRAT supports tracking of two types of markers: *object markers* and *task markers*. Object markers are used for tracking physical objects in AR apps (such as a backpack with supplies in the CRISIS SIMULATION project, cf. Fig. 2). Task markers are used to record events per task (subsequent events will be associated with the task until a new task marker is found, the user says "task completed," or the
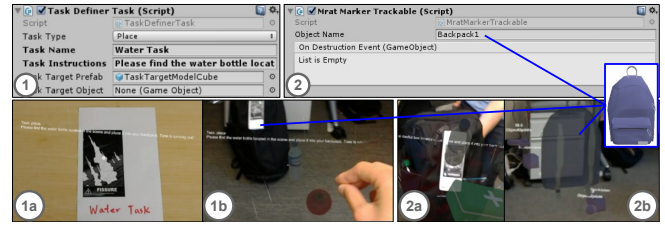

Figure 2. MRAT Task Definition: ① a *Water Task* prompts the user to find the water bottle and place it on a physical, marker-tagged backpack; ② collected items follow the marker on the backpack; later replaying the session will move a virtual backpack along the tracked marker events.

facilitator ends the task in Unity). The mapping from marker to object or task is done in Unity via visual editors provided by MRAT (Fig. 2). While markers are usually physical (patterns printed on paper) and either placed in the environment or handed to participants, MRAT also supports virtual markers. For tracking physical markers, we use Vuforia. When a marker comes into view, a `MarkerFound` event is recorded in MRAT. When Vuforia loses track of that marker, a corresponding `MarkerLost` is recorded. For tracking virtual markers, MRAT can be used to set up gaze listeners that record these events.

Screenshots. MRAT can be configured to take photos at a set interval or automatically when a predefined event occurs, e.g., a tap on a virtual object. MRAT takes photos directly from an available camera, which can be done in MR capture mode on HoloLens or Vuforia-based AR apps to capture the appearance of virtual objects within a user's view (used in our CRISIS SIMULATION & TRIAGE TRAINING app reported later to take pictures of virtual injuries). This is not enabled per default as it can block the MR app's execution (1–2 seconds on HoloLens).

Custom Logging. Finally, we also provide event classes that let MR app developers programmatically define additional events to be logged in custom data structures. Per default, these are stored and visualized like common MRAT tracking data, but when they are logged and how they are visualized can also be changed via script.

2. Task Definition ⇒ *Requirements R4, R5*
The second important process enabled by MRAT is task definition. We developed a generic task concept for MRAT to instrument an MR app with different tracking features and segment collected interaction data on a per-task basis. Tasks with MRAT are defined using visual editors in Unity, requiring no programming efforts (Fig. 2).

Task Specification
Using visual editors in Unity, an evaluator can specify a sequence of tasks to be performed during user testing (Fig. 2). Each task is specified in terms of four main properties:

1. **Task type**, which specifies the action to be performed. By default, this can be a user action, e.g., a *navigate* or *place* task (Fig. 2), a system condition like *creating*, *placing*, or *destroying* a virtual object, or a *custom* task.

2. **Instructions** informing the user what actions need to be performed to complete the task. These can be automatically generated for all non-custom task types.

3. **Target object** or **marker**, specifying a class or instance of a virtual object or fiducial marker as the target of a user action, e.g., for *navigate* and *create* tasks.

4. **Start & end markers** which, upon detection, automatically start and end the task, respectively.

This task concept is flexible and extensible, yet provides relatively simple defaults so that evaluators can get started with specifying tasks in a matter of minutes. We kept the notion of a task as general as possible to support the variety of needs expressed by the interviewed teams:

For experiments similar to Teams PAIN INDICATION, LANDSCAPE MODELING and TRIAGE TRAINING, we developed the four default task types: *navigate*, *create*, *place*, and *destroy*. These represent a class of mechanical tasks common to many MR apps for which MRAT can automatically detect task completion. The *navigate* task can be used to monitor user movements based on the MR app's virtual camera position. In VR, the *create*, *place*, and *destroy* tasks can be used to track both user behaviors and virtual objects present in the scene. Additionally, in AR, the *create* and *place* tasks can be used to track real-world physical objects tagged with fiducial markers. Finally, a *custom* task can be used to implement new Unity scripts to determine task start/end conditions as well as success or failure.

As is shown in Fig. 2, a common use case is to print the marker on a task card that is handed to participants in a usability testing study. Task markers could also be placed in an environment where they can be detected by MRAT. Once a start marker is detected, the participant sees the instructions. When an end marker is detected, the task is recorded as completed and the instructions are hidden. Task completion can always be manually triggered by saying "task completed" or via MRAT's Unity components.

Heuristics & Metrics
To track task completion, we implemented a set of heuristic techniques for each of the mechanical task types in MRAT and a set of metrics to determine task success. In *navigate* and *place* tasks, the user is expected to walk to, or place an object at, a specified target location. The location can be specified in terms of a virtual object in the scene or a fiducial marker placed in the real world. In a *create* task, the user is expected to insert a specified virtual object into the scene or bring a marker-tagged physical object into view. In a *destroy* task, the user needs to remove a specified object or marker. To determine success, MRAT monitors the scene for objects being instantiated and destroyed and listens for `MarkerFound` and `MarkerLost` events, then checks against the specified object or marker. In addition, MRAT determines accuracy as the distance between the user's or virtual object's location and the specified target's location.

We developed these heuristics as a means of abstracting tasks commonly mentioned by our interviewed teams. For instance, Team LANDSCAPE MODELING asked users to plant, move, and remove virtual trees in planned landscape design to control biodiversity. These tasks directly map to *create*, *place*, and *destroy* tasks in MRAT. Team CRISIS SIMULATION asked participants to select from a set of objects—water bottles for drinking and sanitation, food, radios, first aid kits, flashlights, local maps, etc.—and to put together a basic emergency supply kit. This is a task for which there are clearly defined metrics (one gallon of water per person per day for at least three days, at least a three-day supply of non-perishable food, etc.), given the natural limits of what can fit in and be carried in a backpack. As illustrated in Fig. 2, it can easily be mapped to a task sequence in MRAT. Team TRIAGE TRAINING wanted to implement the Simple Triage And Rapid Treatment (START) protocol and its variation, JumpSTART, for use with children. In MRAT, the team wanted to be able to specify tasks that map different triage steps in these protocols for checking respiration and pulse with metrics (upper and lower bound respiration rates, palpable pulse) to assign triage categories (*green*, *yellow*, *red*, *black*). Since the protocols are essentially decision trees, they can be represented in conditional tasks using metrics for dynamically determining task sequence.

3. Session Inspection ⇒ *Requirements R1, R5*
The third and final process enabled by MRAT is session inspection through flexible data visualizations. MRAT functions as an experimentation platform for different visualizations fitting a particular MR app's needs and offers two endpoints for the extension. First, the *MRAT Dashboard* is a web interface implemented using D3.js for *2D Visualizations*. Two default visualizations show the interaction data in a timeline and a top-down floorplan view side by side. Second, the MRAT Unity Package comes with a set of *3D Visualizations* that will be rendered on top of the MR app's content. The default visualizations show the interaction data in MR *in situ*: events will be rendered at the very location at which they occurred, and their visual representation is driven by the event type (e.g., `StatusUpdate` and `ObjectUpdate` events are arrows pointing in the direction of the user's and object's orientation).

The MRAT Dashboard can be accessed from a laptop or tablet, while the MR app can be run in parallel on an AR/VR device. This is particularly powerful when developing an AR app for HoloLens or ARKit-/ARCore-enabled mobile devices. The data can then be inspected with both interfaces being synchronized in real-time, so that the two types of 2D and 3D visualizations can be used in combination. The dashboard implements automatic playback and a manual sliding window to replay sessions as they unfolded over time. Overall, the two types of visualizations, together with global metrics, real-time synchronization, playback, and different means for filtering and selecting individual or sets of events, act as a toolbox from which evaluators can build the visualization they need.

MRAT Dashboard
The 2D visualizations in the MRAT Dashboard enable data exploration based on visual summaries and statistics of all events recorded in a session (Fig. 3). The top part of the dashboard provides information about the individual users ① and modes ② in a session, followed by events ④ and a set of global metrics ⑦ calculated for the current selection of events. All events are visualized on a timeline ⑤ as well as in a floor plan ⑥ of the session, using timestamps and coordinates, while
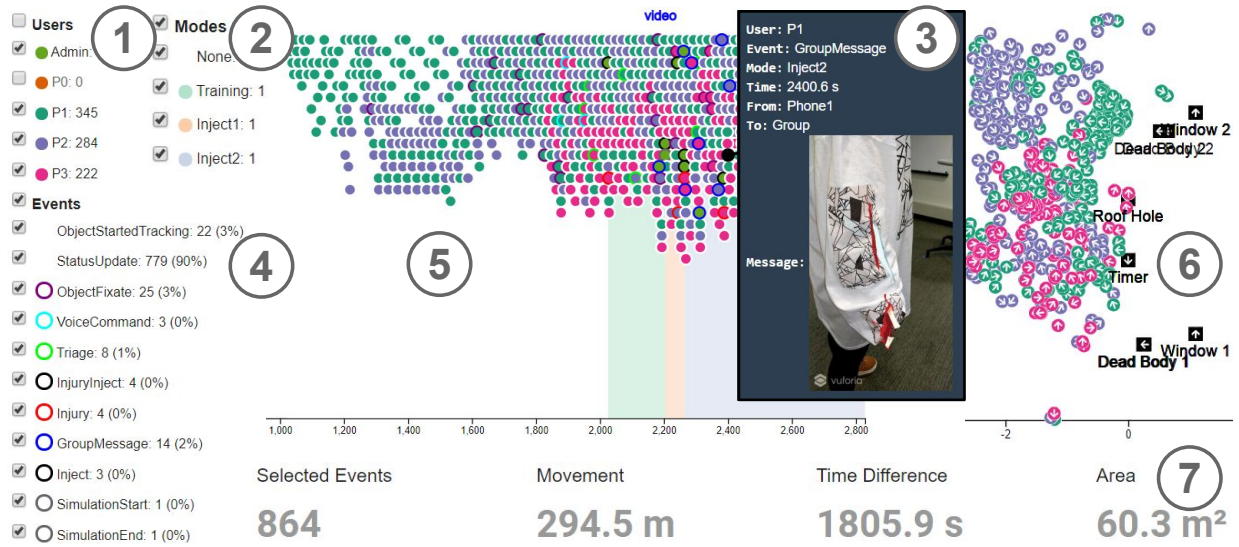
Figure 3. MRAT Dashboard showing a session created when recording the video accompanying the paper: ① events recorded for each user, ② modes, ③ selected event's info (here an injury photo), ④ event statistics, ⑤ timeline and ⑥ floor plan visualizations, ⑦ metrics computed for the selected events.
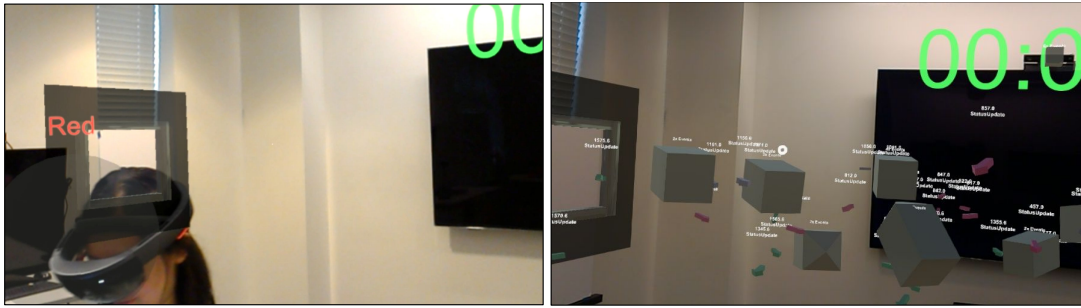

Figure 4. Screenshot on HoloLens from our MRAT-CRISIS case study; in-situ visualizations on HoloLens of the session (same as Fig. 3)—the virtual window and countdown are visible and tracked users and associated injury and triage events are rendered at the captured physical location.

modes are displayed as background colors in the timeline. It is also possible to select an individual event to see details ③.

Filtering. There are five ways to filter events, all of which can be combined: ① selecting certain users; ② selecting certain modes; ④ selecting certain event types; ⑤ brushing the timeline; and ⑥ brushing the floor plan. When combining filters, the result is the intersection of the individual filters. Event statistics ④ and metrics ⑦ are automatically recalculated whenever a filter is applied.

Metrics. The MRAT Dashboard provides a default set of global metrics ⑦ that is automatically calculated from the currently selected events. This set of metrics has been informed by our Requirements Elicitation (i.e., KPIs that were described as useful by multiple AR/VR project teams) and is intended to help evaluators get a quick, high-level overview of an MRAT session. Currently, we provide: the number of *successful, failed,* and *skipped* tasks; the *average task completion time*; the *number of selected events*; *movement*, i.e., the distance the user moved over the selected events; *time difference*; and the *area* covered by the user movement. Inspired by [28], MRAT also implements common interaction metrics: *gesture duration* (how long the user's hand was tracked); *gestures without tap* (or *missed taps*); *taps without a target*; and the user's *activity* in terms of these explicit interactions per second.

In-situ Visualizations

Fig. 4 shows MRAT's in-situ visualizations on top of the running MR app. In-situ visualizations are supported in both VR, using an occluded display like a Windows Mixed Reality headset, and AR, using the HoloLens or an ARCore phone. If an AR device is used in the same environment where the data was collected, the events will appear in the exact real-world locations where they originally occurred, allowing the user to revisit a recorded session exactly the way it happened. However, displaying all data at once often causes events to occlude each other and can lead to poor performance, especially on standalone devices like HoloLens. To mitigate these issues, we implemented interactive, hierarchical clustering techniques that group together dense regions of events into a single 3D marker. This marker is colored, labeled, and sized to indicate the number and type of events it contains. The level of detail is controlled by the user's distance from the centroid of the cluster, such that walking towards markers will ungroup them to show more of the individual events, while walking away from markers will cause them to join with their neighboring markers. Users can also use HoloLens air-tap gestures or a VR controller to interact with groups of events and show more detail in a specific area of interest from their current position.

IMPLEMENTATION & PERFORMANCE

To demonstrate generalizability and account for the variety of AR/VR platforms targeted by our interviewed teams, we implemented the MRAT components for HoloLens, Windows Mixed Reality, and ARCore using Unity and web technologies.

Implementation. The MRAT Unity package is implemented on top of the *Microsoft Mixed Reality Toolkit* (MRTK), with all scripts written in C#. The main components (labeled in blue in Fig. 1) are Unity prefabs that can simply be attached to an existing MR app's virtual camera to start data collection. MRAT's `Trackable` component can be attached to any object in the MR app's scene graph and configured for tracking via MRAT. As part of MRAT's task definition, marker tracking is implemented using *Vuforia* [1]. Two prefabs are available for app instrumentation: `TaskManager` and `TaskItem`. The first enables evaluators to define the test procedure via task sequences and monitor the status of all tasks during the evaluation process. The second is responsible for the specification of each individual task in terms of task type, instructions, target objects/markers, and start/end markers. While MRAT's task definition process requires access to the MR app's scene graph, it usually does not require changing code in existing scripts.

The MRAT Server is implemented based on *Node.js* and *MongoDB*. The server provides session data to the MRAT Dashboard, which is implemented using *jQuery* and *D3.js*. When the selection of events is changed in the Dashboard's timeline or floor plan visualizations, the new set of data is sent to the server via *Ajax* requests. The server then modifies two endpoints (one for checking whether there has been a change and the other providing the new selection) that are long-polled by connected MR apps to feed the 3D visualizations and enable synchronized, *in-situ* data exploration.

Performance. A key implementation challenge was to ensure MRAT has minimal impact on user-noticeable system performance. User interactions are logged by generating MRAT events and placing them into a thread-safe `ConcurrentQueue` data structure processed in Unity coroutines. This approach allows hundreds of events to be sent per second, in addition to full-resolution photos (2048×1152 pixels on HoloLens), without causing a slowdown in the main rendering thread.

We conducted two main performance assessments testing the instrumentation on a variety of MR platforms (HoloLens, Windows Mixed Reality, ARCore) based on 9 open-source apps available from GitHub and stress testing in simulations with more than 9000 continuously tracked virtual objects. Instrumentation with MRAT required less than 20 minutes for 4 apps, 20–30 minutes for 3, more complex apps, and was not possible for 1 app due to its Unity version being older than MRAT's minimum supported version (2017.1.0p5).

Our stress testing revealed that MRAT's defaults for the user and object tracking, and even when taking screenshots, achieve high performance when continuously tracking a large number of virtual objects (increased in steps of 100 after every minute, up to 9500). We created a synthetic performance metric to allow direct comparison between the different platforms. The goal was to achieve an average FPS of $\geq$59 ($\geq$29 with AR-Core, due to the platform not supporting 60 FPS). We found that MRAT was able to track 1.1k objects on HoloLens at 60 FPS without screenshots (50 FPS with screenshots), 6.5k objects on ARCore at 27 FPS (still 6.5k objects at 22 FPS with screenshots), 8k objects on Windows MR at more than 60 FPS (7.8k objects with screenshots), and 9.5k objects on the same PC just in Unity without VR (8.7k objects with screenshots).

OPEN-SOURCE RELEASE

MRAT and detailed results of our performance tests are available at `https://github.com/mi2lab/mrat`.

MRAT-CRISIS CASE STUDY

While developing MRAT, we collaborated with four of the interviewed teams (FILM, LANDSCAPING, CRISIS, TRIAGE), identifying several use cases for MRAT through workshops with them. While all workshops were interesting and indicated that MRAT could be applicable in many projects, we present one example in detail as our case study. This section reports on the use of MRAT to support the design and evaluation of MRAT-CRISIS, a new MR app we created in collaboration with CRISIS SIMULATION and TRIAGE TRAINING teams.

The entire research reported here, including the development of both MRAT and MRAT-CRISIS, covered a period of 18 months. The core of MRAT was completed over the first 8 months of research. The development of the new MR app required another 8 months. The final two months were spent on using MRAT for testing and tuning the MR app. The last two weeks were focused on study design and MRAT configuration in terms of tasks and metrics for user testing.

Below, we first describe the design process of MRAT-CRISIS. In particular, we highlight the use of MRAT for system performance tuning. We then report on MRAT-CRISIS testing sessions with students and instructors. We see this both as a blueprint for using MRAT in user studies and an evaluation of the use and usefulness of MRAT to support such studies.

Requirements & Design Process

The basis of the CRISIS SIMULATION and TRIAGE TRAINING projects were classroom exercises. To better understand requirements for translating these experiences to MR, we participated in classroom observations and a triage training exercise.

Crisis Simulation Classroom Exercise

We observed two teams of eight and nine students over a one-hour exercise developed by a crisis informatics instructor. Students were instructed to be creative, yet realistic, in solving a disaster scenario consisting of four timed injects, which trigger events (such as a mortar injuring a random selection of players) and force players to adapt to changes in their situation. To make for an immersive exercise (without using MR yet), the simulation started with the *Omaha Beach* scene from the film *Saving Private Ryan* playing on eight TVs around the classroom. Teams received a map to a remote "headquarters" (HQ) location and tasked with sending one player (the runner) to HQ when it was safe. An assistant supervising the exercise manually recorded the runner's departure time and announced injects. Once the runner left the room, a second assistant stationed outside the classroom monitored their activity.

While observing the exercise, we identified areas of the current simulation which could benefit from MRAT's capabilities. One runner left the room before it was safe (the student seemed unaware of ongoing mortar shelling) and the other runner improvised their route to HQ without the hall monitor's knowledge. We saw an opportunity to map the runner's route with MRAT and assess task success in terms of the exact departure and arrival times.

Tabletop Triage Training Exercise
To further guide the design of our MR simulation, we participated in a tabletop triage training exercise led by a nursing professor. Presented with a train-crash scenario, four of the researchers worked together in pairs to assess civilians (represented by Lego figures and index cards with their injury information) and assign them triage labels in terms of the START and JumpSTART protocols. After categorizing all patients, we decided when, to which hospital, and with which mode of transportation to send patients for medical care.

The tabletop exercise informed, in particular, the design of our triage tasks, as we better understood the flow of communication in a crisis situation and what information is important to convey to the incident commander. We saw an opportunity to enhance the realism and increase the level of stress in this theoretical triage exercise through projecting injuries in MR.

T-shirt designs
In order to display five types of injuries varying in bodily placement and severity, we designed t-shirts with Vuforia markers. While Vuforia marker tracking is fairly robust in many settings, our goal was to show realistic injuries in a dark physical environment and in anatomically-correct locations on the players, and this proved to be difficult for marker tracking. Therefore, we began with paper prototypes to test the performance of markers in terms of shape, location, and size. We then used transfer sheets to iron markers on both white and black shirts, comparing the realism of injury projections and marker tracking performance with both colors. (Fig. 5).
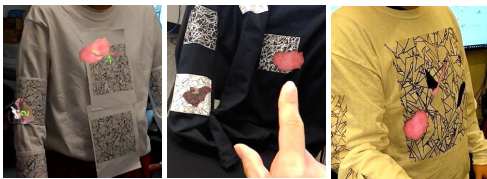

Figure 5. Three t-shirt versions from left to right: V1—with placeholder markers at different locations and sizes; V2—with ironed-on markers, in black and white; V3—with large, black markers using professional shirt printing for front markers and ironed-on markers for sleeves.

Implementation
Figure 6 shows the system architecture of our MRAT-CRISIS implementation: it consists of an *Admin Client* (implementing messaging and simulation control features) operated by a lab assistant in control of the simulation, as well as pairs of *Phone Clients* and *HoloLens Clients* (implementing messaging, inject, injury, and triage features) used by each player.

The admin controls the start and duration of the simulation (indicated by a countdown timer visible to the admin and players), as well as issues injuries and injects to advance the
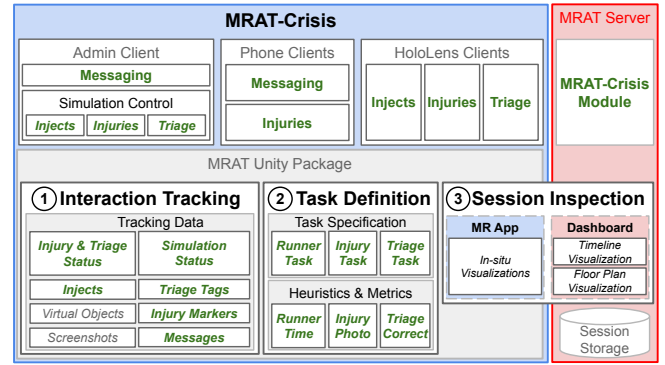

Figure 6. Architecture of our MR app after instrumentation with MRAT: The MR app consists of an *Admin Client* to send messages to players and control the simulation, *Phone Clients* for taking and sharing photos of injuries, and *HoloLens Clients* adapting the MR environment based on injects, superimposing injuries on players' marker-labeled t-shirts, and supporting triage via voice commands. We configured MRAT to track data specific to the simulation and tasks, defined three tasks with a simple metric for each, and implemented a module for the MRAT Server to synchronize the simulation between all clients.

stage of the simulation. They also assign the medic role to players who will perform triage and deliver instructions to players' phones via a group chat.

Each player is equipped with a phone and a HoloLens. Through the phone, players receive instructions sent by the admin, chat with the group, and use the camera to send photos of the injuries superimposed on players' shirts. The HoloLens processes the injects received from the admin, including a mortar explosion and storm, by changing the MR environment. When the admin assigns injuries to a player, their HoloLens, like the phone, shows images of wounds on top of the Vuforia markers on the players' shirts. Players in the role of medic must focus on another player's head and issue voice commands ("green" for minor injuries, "yellow," "red," or "black" for increasingly severe ones) to assign triage categories.

To facilitate the multi-user, cross-device crisis simulation, we developed a module for the MRAT Server which facilitated admin actions (simulation start and end, delivery of injects and injuries, messaging phone clients) and phone actions (messaging the admin or other phone clients). While extending the server is *not* a requirement for integrating new MR apps with MRAT, we did this to manage all simulation data centrally.

Instrumentation with MRAT
Figure 6 also illustrates how we instrumented MRAT-CRISIS, making use of MRAT's various components with minor code extensions. We used MRAT's interaction tracking capabilities to capture players' fixations on the timer (as a proxy for stress level), injects issued by the admin and the corresponding stages of the simulation, triage labels assigned by players, as well as messages sent and photos taken by players.

To address specific requirements of MRAT-CRISIS, we benefited from MRAT's various extension points. We extended the `Trackable` behavior of MRAT to include `ObjectFixate` events, which are reported when players gaze upon tracked objects for a configurable period of time (1s per default). For the purposes of our simulation, we only tracked timer fixa-

tions. We created custom `StatusUpdate` events for *Injury*, to record the injuries a user receives, and for *Triage*, to record the triage tags assigned by the medic. We also extended MRAT's marker tracking to log `MarkerFound/Lost` events for the injury markers. Finally, rather than logging individual modes in each client, we reported the simulation status maintained by the MRAT-CRISIS Server module through mapping injects to modes in MRAT. We then used the MRAT-CRISIS Server module to log messages sent by the admin and phone clients.

All of the above extensions are now part of MRAT's default feature set; MRAT-CRISIS informed the design of parts that could not be derived from the Requirements Elicitation alone.

Performance Tuning using MRAT

Apart from enabling user testing, MRAT also played an important role in various design decisions and system performance, which is a crucial usability factor, of MRAT-CRISIS. We faced two main challenges: *1)* keeping the FPS high while also making for an immersive experience on HoloLens with realistic injects, and *2)* keeping injury marker tracking stable on HoloLens and phone clients with dimmed lights in the room.

Optimizing the HoloLens Experience

To optimize the design of the MR world, we utilized the system performance monitoring feature of MRAT. Initially, we wanted to include a mortar explosion with smoke particles, but the asset was extremely slow to render on HoloLens. We used MRAT to measure the system performance with different variations of the asset, tracking FPS readings and issuing `PerformanceWarning` events when the frame rate dropped below 30 FPS. With the initial version of the asset, the frame rate was as low as 18 FPS on average. We achieved the best performance for the mortar inject, 46 FPS on average, by placing holograms far away from the viewer and lowering the particle count and size, leading us to design the MR world such that a majority of assets were placed outside virtual windows where they would only need to be partially rendered.
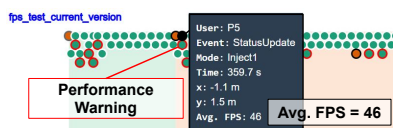

Figure 7. FPS tests of final MR world design: inspecting critical states flagged by `PerformanceWarning` events guided our optimizations; we achieved 46 FPS on average for the most immersive inject (mortar).

Two additional issues we faced were HoloLens dropping frames with continuous spatial mapping and losing tracking in a large room full of people, especially with dimmed lights. We were able to detect and quantify these issues in MRAT sessions via `PerformanceWarning` and significant positional changes between `StatusUpdate` events (the HoloLens camera "freezes" and temporarily sets the player's position near $(0, 0, 0)$ when tracking is lost). However, we were unable to resolve the issues in this first prototype of MRAT-CRISIS and decided they were out of scope. We limited spatial mapping to be performed in an empty room prior to user testing, and instead focused on making the MRAT-CRISIS server module stable to re-sync clients after quick recalibration.

Optimizing the T-shirt Designs

To optimize the t-shirt designs and achieve stable Vuforia marker tracking with spatial mapping in a dimmed environment, we recorded sessions with MRAT comparing the black and white marker designs for two different light conditions in the room we aimed to use for user testing (Fig. 8). With all five injuries activated for the markers, we then performed the same set of movements for each shirt and condition: scanning the shirt top-down and bottom-up, from either side, close-up and from afar. In the best case, markers were recognized right at the start of a session and tracked continuously throughout. The data indicated the white shirt with lights on performed best (as none of the markers were tracked continuously, sessions with more `MarkerFound` events indicate better performance).
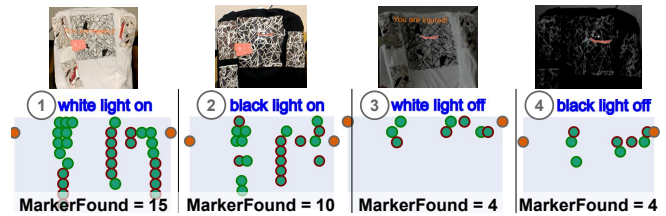

Figure 8. Marker tests: ① white shirt with lights on performed the best (15× `MarkerFound` events); ② black shirt with lights on was second best (10×); ③ and ④ either shirt with lights off performed worst (4×).

User Testing Sessions of MRAT-CRISIS

In our user testing sessions of MRAT-CRISIS, the goal was two-fold: we wanted to get feedback from students on the user experience compared to the original classroom exercise, and from the instructors, as evaluators of the students' performance in the simulation, on the various MRAT visualizations:

- Can instructors tell from the various visualizations (timeline, floor plan, in-situ) how the simulation unfolded without actually being in the room?

- How would instructors go about analyzing the students' behaviors during the simulation? Can they explain what happened from the data?

- What are useful metrics to assess task success? How would they compare the performances of two student teams and determine which one did "better?"

Procedure

With the help of the instructors, we recruited two teams of 5–6 students each (*N*=11, 8 female, three male, average 24 years of age) who had also participated in the classroom exercise we observed. We provided students with a HoloLens, phone, and t-shirt with Vuforia markers, then instructed them to complete three tasks during the simulation and a questionnaire afterward. One member of the research team acted as a lab assistant to control the simulation via the admin client.

Similar to the original classroom exercise, the two instructors were out of the room for the duration of the experiment. Unlike the original experience, where the only means of learning about the simulation stage was by messaging the lab assistant, the instructors were given access to the MRAT Dashboard to track the simulation progress remotely. After the simulation, the instructors returned to the room, where we gave them each

a HoloLens with the in-situ visualizations, in addition to the timeline and floor plan in the dashboard, and asked them to use the visualizations to make inferences about task completion.

Tasks & Metrics

We structured the user test around three major tasks:

1. *Runner task*: sending one player to a remote "headquarters" location—runner must notify the group of their safe arrival;

2. *Injury task*: sending group messages with photos of all injuries of all players who were injured;

3. *Triage task*: assessing the severity of players' injuries and assigning them a triage tag (green/yellow/red/black).

The admin communicated all tasks with participants via messages to the group chat. The runner task was concurrent with the other tasks in that participants had to determine the "right time" to leave the safe house. At the start of the simulation, the first inject (mortar) inflicted injuries on a random selection of players. This triggered the injury task, followed by the triage task. The simulation ended when either the runner made it to headquarters safely or the 10-minute countdown had expired.

For the purpose of testing this first version of MRAT-CRISIS, we defined three basic metrics for analysis:

1. *Runner Time*: select the region on the floor plan that contains the farthest out `StatusUpdate` events and pick the time from the earliest event in that region;

2. *Injury Photo*: select all `GroupMessage` events and filter down to those with a preceding `MarkerFound` but no `MarkerLost` event;

3. *Triage Correct*: select all `Triage` events with preceding `Injury` events, and compare against a look-up table with the correct triage tags.

Findings

In terms of the metrics, both groups finished the simulation in under 10 minutes. The first group's runner left early (shelling was still ongoing), and was labeled as deceased. The rest waited until the storm started and left the room together in a group. This was against the rule of having one runner, but since it was technically safe to leave we recorded this as *Runner Time*$_1$=571 s. The second group performed better in terms of *Runner Time*$_2$=476.2 s. Only the first group sent photos of the injuries (*InjuryPhoto*$_1$=2), the second group misread the task (*InjuryPhoto*$_2$=0). Both groups, however, applied the correct triage tags (*TriageCorrect*$_1$=*TriageCorrect*$_2$=2).

When analyzing the session Dashboards, the instructors found the visualizations enabled them to imagine what happened during the simulation. They used metrics similar to ones defined by us and found it easy to identify the runner using the floor plan visualization by "finding the dot furthest to the left" (physically most separate from the rest of the group). When asked to identify who was injured, they expressed they would first look at participants who stopped moving until the runner returned. Both instructors used their intuition rather than looking for the specific `Injury` events generated by MRAT-CRISIS, which suggests they were confident enough to make

inferences just from visualizations, even if not everything was captured directly in terms of events.

We also asked what the instructors might look for as metrics of success when comparing the performance of the two groups. They agreed there is not an easy-to-spot, objective indicator of success, but there is a crucial step after an initial period of chaos when people begin to self-organize by assigning roles. One way they would approach finding this "threshold of organization" based on the visualizations would be to "connect dots" representing people who stay close together for a period of time. When we explained to the instructors how we could add a social network analysis metric to the visualizations, they expressed that this would be "incredibly powerful." They also stressed that the visualizations would benefit from annotations, e.g., for labeling players to indicate role selection.

Finally, the players' feedback questionnaire responses were mostly positive and encouraging, but from our observations, during the simulation, we identified some usability issues. Players from the first group felt that "only seeing one person's injuries at a time" and "not knowing the vocal interface of triage commands" slowed them down. From the `MarkerFound/Lost` events, we could see that the markers were not always consistently shown to all looking at the injured players. We observed that players stood closely together, perhaps leading to marker occlusions. However, the data suggests that we need to further improve the t-shirt designs.

One participant noted "some disturbances in visual" and stated that "the environment was very blurry... I was distracted by the artifacts in the room." This feedback is hard to quantify, but we inspected the data and found several shifts in the `StatusUpdate` recorded for both groups. When shifting occurs, the augmentations of the room become misplaced, leading to a major disruption of the suspension of disbelieve (and the user experience). One possible extension would be to build in `TrackingWarnings` that can alert assistants and directly point them to the players experiencing these issues.

CONCLUSION

MRAT presents concepts and techniques designed to support usability evaluations of MR apps, through plug-in-like instrumentation of the apps in Unity, the use of virtual or physical markers to define user tasks, and a set of heuristics and metrics that can feed visualizations for analysis. Limitations of our work include our focus on one case study, MRAT-CRISIS, to assess the use of MRAT, and the proof-of-concept nature of the studies conducted with it. Our main goal was to assess the feasibility of MRAT for usability evaluations, not testing the usability of MRAT or its effectiveness for using MR in this type of educational simulation. The generalizability of MRAT still needs to be demonstrated in summative evaluation, but we aimed to address requirements from a variety of domains via the formative interviews. Future research should study how other researchers may use and extend tools like MRAT to fit their needs, through studies on toolkit usability. It will also be interesting to study how tracking capabilities of new devices like HoloLens 2 (e.g., finger tracking and actual gaze) can be leveraged in MR evaluations with future versions of MRAT.

REFERENCES

[1] 2018. Vuforia. (2018). `https://www.vuforia.com`.

[2] Ayman Alzayat, Mark Hancock, and Miguel A. Nacenta. 2019. Quantitative Measurement of Tool Embodiment for Virtual Reality Input Alternatives. In *Proc. CHI*. 443.

[3] Practical Analytics. 2017. Practical Analytics for HoloLens. (2017). `https://analytics.practicalvr.com`.

[4] Daniel Ashbrook and Thad Starner. 2010. MAGIC: A Motion Gesture Design Tool. In *Proc. CHI*.

[5] Richard Atterer, Monika Wnuk, and Albrecht Schmidt. 2006. Knowing the User's Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In *Proc. WWW*.

[6] Zhen Bai and Alan F Blackwell. 2012. Analytic review of usability evaluation in ISMAR. *Interacting with Computers* 24, 6 (2012), 450–460.

[7] Mark Billinghurst, Adrian Clark, and Gun Lee. 2015. A Survey of Augmented Reality. *Foundations and Trends® in Human–Computer Interaction* (2015), 73–272.

[8] Tonio Carta, Fabio Paternò, and Vagner Figuerêdo de Santana. 2011. Web Usability Probe: A Tool for Supporting Remote Usability Evaluation of Web Sites. In *Proc. INTERACT*.

[9] Microsoft Windows Dev Center. 2019. Understanding Performance for Mixed Reality. (2019). `https://docs.microsoft.com/en-us/windows/mixed-reality/understanding-performance-for-mixed-reality`.

[10] Cognitive3D. 2017. Hololens Analytics. (2017). `https://cognitive3d.com/blog/hololens-analytics`.

[11] Arindam Dey, Mark Billinghurst, Robert W. Lindeman, and J. Edward Swan II. 2018. A Systematic Review of 10 Years of Augmented Reality Usability Studies: 2005 to 2014. *Front. Robotics and AI* 2018 (2018). DOI: `http://dx.doi.org/10.3389/frobt.2018.00037`

[12] Andreas Dünser and Mark Billinghurst. 2011. Evaluating augmented reality systems. In *Handbook of Augmented Reality*. 289–307.

[13] Andreas Dünser, Raphaël Grasset, and Mark Billinghurst. 2008. *A Survey of Evaluation Techniques Used in Augmented Reality Studies*. Technical Report. Human Interface Technology Laboratory, University of Canterbury.

[14] Andreas Dünser, Raphaël Grasset, Hartmut Seichter, and Mark Billinghurst. 2007. *Applying HCI principles to AR systems design*. Technical Report. Human Interface Technology Laboratory, University of Canterbury.

[15] Eivind Flobak, Jo Dugstad Wake, Joakim Vindenes, Smiti Kahlon, Tine Nordgreen, and Frode Guribye. 2019. Participatory Design of VR Scenarios for Exposure Therapy. In *Proc. CHI*. 569.

[16] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. 2007. Direct-Touch vs. Mouse Input for Tabletop Displays. In *Proc. CHI*.

[17] Adam Fouse, Nadir Weibel, Edwin Hutchins, and James D. Hollan. 2011. ChronoViz: A System for Supporting Navigation of Time-coded Data. In *Proc. CHI EA*.

[18] Qi Guo and Eugene Agichtein. 2010. Ready to buy or just browsing?: detecting web searcher goals from interaction data. In *Proc. SIGIR*.

[19] Uta Hinrichs and Sheelagh Carpendale. 2011. Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits. In *Proc. CHI*.

[20] Sujin Jang, Niklas Elmqvist, and Karthik Ramani. 2014. GestureAnalyzer: Visual Analytics for Pattern Analysis of Mid-Air Hand Gestures. In *Proc. SUI*.

[21] Jaeyeon Lee, Mike Sinclair, Mar González-Franco, Eyal Ofek, and Christian Holz. 2019. TORC: A Virtual Reality Controller for In-Hand High-Dexterity Finger Interaction. In *Proc. CHI*. 71.

[22] Zhen Li, Michelle Annett, Ken Hinckley, Karan Singh, and Daniel Wigdor. 2019. HoloDoc: Enabling Mixed Reality Workspaces that Harness Physical and Digital Content. In *Proc. CHI*. 687.

[23] Blair MacIntyre, Maribeth Gandy, Steven Dow, and Jay David Bolter. 2004. DART: a toolkit for rapid design exploration of augmented reality experiences. In *Proc. UIST*.

[24] Paul Milgram and Fumio Kishino. 1994. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems* 77, 12 (1994), 1321–1329.

[25] Michael Nebeling, Fabrice Matulic, and Moira C. Norrie. 2011. Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In *Proc. CHI*.

[26] Michael Nebeling, David Ott, and Moira C. Norrie. 2015. Kinect Analysis: A System for Recording, Analysing and Sharing Multimodal Interaction Elicitation Studies. In *Proc. EICS*.

[27] Michael Nebeling and Maximilian Speicher. 2018. The Trouble with Augmented Reality/Virtual Reality Authoring Tools. In *Proc. ISMAR Adjunct*.

[28] Michael Nebeling, Maximilian Speicher, and Moira C. Norrie. 2013. W3Touch: Metrics-based Web Page Adaptation for Touch. In *Proc. CHI*.

[29] Jeffrey Nichols, Zhigang Hua, and John Barton. 2008. Highlight: A System for Creating and Deploying Mobile Web Applications. In *Proc. UIST*.

[30] J. Nielsen. 1999. *Designing web usability: The practice of simplicity*.

[31] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic Books (AZ).

[32] Oculus. 2019. Guidelines for VR Performance Optimization. (2019). `https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-performance-guidelines`.

[33] Ken Pfeuffer, Matthias J. Geiger, Sarah Prange, Lukas Mecke, Daniel Buschek, and Florian Alt. 2019. Behavioural Biometrics in VR: Identifying People from Body Motion and Relations in Virtual Reality. In *Proc. CHI*. 110.

[34] Valentin Schwind, Pascal Knierim, Nico Haas, and Niels Henze. 2019. Using Presence Questionnaires in Virtual Reality. In *Proc. CHI*. 360.

[35] Emily Shaw, Tessa Roper, Tommy Nilsson, Glyn Lawson, Sue V. G. Cobb, and Daniel Miller. 2019. The Heat is On: Exploring User Behaviour in a Multisensory Virtual Environment for Fire Evacuation. In *Proc. CHI*. 626.

[36] Maximilian Speicher, Brian D. Hall, and Michael Nebeling. 2019. What is Mixed Reality?. In *Proc. CHI*.

[37] Maximilian Speicher, Brian D. Hall, Ao Yu, Bowen Zhang, Haihua Zhang, Janet Nebeling, and Michael Nebeling. 2018. XD-AR: Challenges and Opportunities in Cross-Device Augmented Reality Application Development. *PACMHCI* 2 (2018), 7:1–7:24.

[38] Zhida Sun, Feng Han, and Xiaojuan Ma. 2018. Exploring the Effects of Scale in Augmented Reality-Empowered Visual Analytics. In *Proc. CHI EA*.

[39] Hemant Bhaskar Surale, Fabrice Matulic, and Daniel Vogel. 2019. Experimental Analysis of Barehand Mid-air Mode-Switching Techniques in Virtual Reality. In *Proc. CHI*. 196.

[40] J E Swan and J L Gabbard. 2005. Survey of User-Based Experimentation in Augmented Reality. In *Proc. International Conference on Virtual Reality*.

[41] Yuki Tayama, Ryuga Kato, and Ken-ichi Okada. 2015. Triage training system: adjusting the difficulty level according to user proficiency. In *Proc. MUM*. 139–147.

[42] Unity Technologies. 2017. Unity Analytics. (2017). `https://unity3d.com/solutions/analytics`.

[43] Alexandra Voit, Sven Mayer, Valentin Schwind, and Niels Henze. 2019. Online, VR, AR, Lab, and In-Situ: Comparison of Research Methods to Evaluate Smart Artifacts. In *Proc. CHI*. 507.