

# A foray into the use of serious games in controlled research on crisis management

**Michael E. Stiso**  
SINTEF ICT  
michael.stiso@sintef.no

**Aslak W. Eide**  
SINTEF ICT  
aslak.eide@sintef.no

**Antoine Pultier**  
SINTEF.ICT  
antoine.pultier@sintef.no

## ABSTRACT

Controlled experiments on crisis management could provide many insights into the human factors that lead to effective performance in the area. However, the challenge of establishing a controlled environment directly relevant to the chaotic settings in which crisis management occurs means that such experiments are scarce. Here, we describe our attempt to use a videogame (ARMA III) as a realistic but controllable environment for research in this domain. We successfully developed a testbed linking the game world to the front-end of a prototype command-and-control system, so that one can use the latter to monitor events in the former. However, when it came to developing controlled scenarios for the experiment, we discovered that too much realism can be a problem. This paper outlines the challenges we encountered and provides recommendations for researchers and game designers interested in the use of serious games in scientific research.

## Keywords

controlled experiments, crisis management, games, research methods, virtual worlds

## INTRODUCTION

Investigating individual and team performance in crisis management via controlled experiments is a challenging task, but also a worthwhile one. Controlled experiments in this area can yield causal insights into the personal, social, and environmental factors behind effective crisis management, as well as whether and how different support tools can influence performance. Unfortunately, this method of research depends on the ability to hold circumstances constant across a set of experiments, so that comparisons can be made regarding the effects of specific variables on measured outcomes. That can be exceedingly difficult to achieve in the context of crisis management work.

Crisis management is a particularly thorny setting because it occurs in dynamic environments characterized by uncertainty, high stress, and tight time constraints. Such situations are difficult to recreate in the lab. The situation is further complicated because those involved in crisis management are highly trained personnel, making them relatively difficult to access for experimentation purposes. As a result, efforts at controlled research in the area usually must involve either

1. an experimental task simple enough for random participants to learn,

complex enough to reveal differences in performance between experimental conditions, and set in a closed, controllable environment.

2. real subject-matter experts performing a domain task that is realistic enough for them to make use of their expertise, but which still allows at least a modest degree of experimental control.

A third alternative may be possible, thanks to the rise of the serious-gaming movement over the last decade. Serious games tend to be used mainly for training and educational purposes (e.g., see Connolly, Boyle, MacArthur, Hainey, and Boyle, 2012). However, with its efforts to create increasingly realistic and customizable virtual game worlds, the videogame industry may also have produced virtual environments capable of serving as realistic yet controllable environments for scientific research on crisis management. This paper reports on our first attempt at creating such an experimental testbed, the challenges we encountered, and feature recommendations for researchers and game designers interested in pursuing serious games for experimentation.

## METHOD

### Strategy

Our group's focus is on the human-machine interfaces (HMI) of crisis management command-and-control (C2) systems, which serve as mediators between users and the external world. Such HMIs present users with dynamic, interactive information pulled from sensors, personnel, and other external sources, possibly alongside decision support tools to help users interpret and act on that information. The data presented can vary with the system, but in crisis management likely includes the location and status of resources and other on-scene entities, perhaps a timeline of events, and virtual windows to the external world via video feeds and images from on-scene cameras.

We had a prototype C2 HMI ready for testing because of our recent participation in the DARIUS<sup>1</sup> EU FP7 project (Figure 1). However, we did not have much

<sup>1</sup> <http://darius-fp7.eu>



Figure 1. A prototype C2 HMI from the DARIUS project.

access to a real-world environment in which to do repeated testing – i.e., a situation from which the HMI could pull sensor data, resource information, and videos. So, we attempted to replace the real world with a commercially available virtual one capable of running participants through scripted and controllable crisis scenarios.

### ARMA III

We selected ARMA III<sup>2</sup>, developed by Bohemia Interactive. ARMA is a tactical, first-person shooter that is set in an expansive, open virtual world (Figure 2). The core component of gameplay is its scenario editor (Figure 3), which allows users to create, play, and share game scenarios, using either artificial intelligence or other online players as opponents. We settled on ARMA for a few key reasons:

- It fit well with our existing custom software components, making it

<sup>2</sup> <http://arma3.com/>



**Figure 2. An aerial view of the virtual world within ARMA III.**

relatively easy (compared to other environments) to exchange data between ARMA and other software applications.

- Its list of ready-made virtual entities includes unmanned vehicles, both air (fixed- and rotary-wing) and ground, which was important because our C2 HMI prototype was geared toward the management of unmanned vehicles.
- It has a sophisticated editor and integrated scripting language, good documentation, a lot of third-party extensions, and a modifiable UI.

The result was a testbed consisting of two parts (see Figure 4): a homemade C2 HMI and a commercially available virtual world (ARMA). We built a link between the two for information exchange, enabling our HMI to (1) display an interactive map of the ARMA virtual island, which was based on the real-world Lemnos in the Aegean; (2) position icons on that map that represent the current

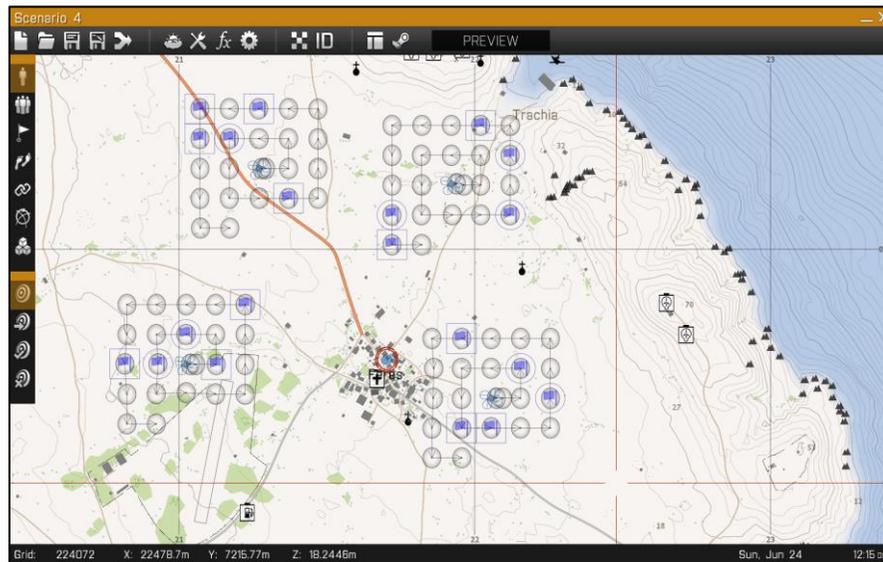
locations and status of entities on the island; and (3) provide access to the camera feeds of selected entities so that users could view those entities' current virtual surroundings. In effect, our C2 HMI provided a view into a virtual world rather than the real one, giving us the opportunity to study the HMI's use in crisis management within a seemingly controllable and easily modifiable environment.

Note that we also considered using Bohemia Interactive's Virtual Battlespace 2 (VBS2) environment. We had been using VBS2 in a concurrent project, although not for scientific research, and so we were familiar with its capabilities. We decided against it for a couple of reasons:

- At the time, the only pre-built drone available in VBS2 was the Predator, a fast- and high-flying fixed-wing aircraft. We needed a low-flying copter drone, however, which ARMA III provides.
- VBS2 comes with example maps, but they did not match the expansiveness and quality of the worlds provided in ARMA III. Users can make their own maps, of course. In our previous use of VBS2, though, we found that building even a very small setting (e.g., a virtual city dock) required a considerable amount of time and effort, and it would never match what we had available in ARMA.
- The process of connecting our HMI to VBS2 was not as straightforward as connecting it to ARMA III.

### Scenario design

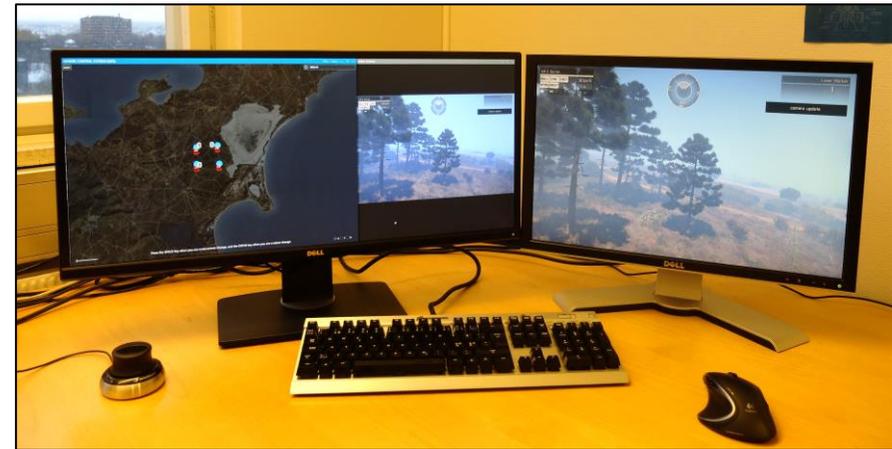
For our pilot project, we set up a simple search-and-rescue test scenario relevant to the aforementioned DARIUS project from which our HMI originated. The task itself was adapted from that in (Parasuraman, Cosenzo, De Visser, 2009). The scenario takes place on an island after a recent terrorist attack. Survivors have fled into the countryside and are being pursued by the terrorists. Drones are flying search patterns across the island to locate both types of individuals, so that rescue teams can reach the former while avoiding the latter. The drones can detect people within a certain range, upon which the test participants must open the drone's camera feed and identify the person (or "target") as hostile or friendly.



**Figure 3.** The ARMA scenario editor, showing four spiral flight paths for the drones. The circles within each path represent waypoints, the flags represent targets. The waypoints are spaced in intervals of 100m.

Building that scenario meant using the ARMA editor to place entities (4 drones and a mix of 20 survivors and terrorists) and objects (flight paths and waypoints) on an interactive map of the virtual island. We then had to modify the appearance of the targets, add triggers to make them visible when a drone got within range, and dumb down the AI of all of the entities to make them more predictable.

When the scenario was running, the ARMA environment would not be directly visible to participants. Rather, they would monitor the four drones via a map on the HMI displaying four moving drone icons. The targets were spread across the drone flight paths, sequenced so that only one target was detectable at a time. Upon detection, that target appeared as an unidentified icon on the map. Participants then had a short window to open the given drone's camera feed and identify the target as a survivor or a terrorist. The feed would appear in a separate



**Figure 4.** The basic experimental setup. The monitor on the left shows the C2 HMI, within which the user has selected a drone. That action opens the drone's camera feed, which can be seen in the smaller window to the right of the HMI. The monitor on the right shows the view from the selected drone flying in the ARMA virtual world. Note that test participants would not see the ARMA monitor; it is shown here just to demonstrate that it and the HMI camera feed show the same thing.

window in the HMI and show the camera view of the drone's current virtual surroundings (including the target).

With the basic scenario concept in hand, we set out to create a series of test scenarios for a controlled experiment. That is when we started to hit obstacles, both literally and figuratively.

### CHALLENGES WITH CONTROL IN VIRTUAL WORLDS

Interestingly, the problems we encountered were analogous to problems we would have encountered performing this experiment in the real world, which in retrospect makes sense given that virtual worlds are an attempt to mimic the real thing. The various issues are grouped into three areas dealing with realism, vehicle control, and precision editing.

Note that the latest version of VBS – VBS3 – addresses some of the issues discussed below, and it even includes a rotary-wing drone. However, VBS3 was not available until after we had already invested significant time in trying to make ARMA III work, and we still would have had the issues with connectivity and pre-built worlds.

### Too realistic

#### *Drone speed*

The drones available in ARMA all had seemingly realistic maximum speeds. While great for gameplay, the type of drone suitable for our scenario did not travel fast enough for a single one (our initial goal) to easily accommodate 20 non-overlapping targets in its flight path within a 6-minute scenario (another goal).<sup>3</sup>

The distance covered in that time would just fit all the targets, with just enough spacing between them to allow 6-8 seconds for identification before the next one was detected. However, that created too much uniformity in target sequencing, enabling test participants to anticipate when and where the next target would appear. That, in turn, would have confounded our results, which were based on reaction time and accuracy. Our workaround was to break from the original design of a single drone and instead spread the 20 targets (a necessary data sample) amongst four drones, making it difficult to guess the next target location.

#### *Drone cameras*

Ideally, we could have had the drone travel a little faster than realism would allow, to accommodate a non-uniform distribution of targets along its path. However, increasing its speed would introduce another realism-related problem: Just as with a real drone, it is difficult to spot and identify people via a camera feed if the virtual drone is moving too quickly.

In addition, if targets were not placed directly within the flight path of a drone,

<sup>3</sup> Note that a recent update to ARMA III increased drone speed considerably.



**Figure 5: The view from a drone. The red character near the bottom is a terrorist. Participants had about 8 seconds to open the camera feed and classify the person.**

they would appear in the periphery of the camera feed. That made them more difficult to identify, both because the extra distance from center made them visually smaller, and because objects in the periphery had less time on camera than did those right in front of the drone. That is good for gameplay, but for a controlled experiment in which all targets should be equally visible and identifiable, it meant all targets had to be front and center. (See Figure 5.)

#### *Drone obstacles*

Our main frustration can be best summed up in one word: Trees. In the real world, trees are a problem for drone health, as are telephone poles and most any other tall ground object. Drones can generally fly high enough to avoid such obstacles, but the higher they fly, the more difficult it becomes for drone operators to locate and identify ground objects in the camera feed. As it turns out, the same issue exists in



**Figure 6: The bane of drones.**

a realistic virtual world (Figure 6). However, unlike real-world drones, the ones in ARMA did not have cameras with zoom capabilities. For the targets in our scenarios to be visible in the camera feeds, then, the drones had to fly either in a treeless area or in a hazardous route through the treetops.

Unfortunately, there were no treeless areas on the ARMA island big enough to accommodate four drones flying adjacent flight paths. Consequently, we had to route each one through tree-laden areas, which required a trial-and-error process involving running the scenario, noting when a drone crashed, altering its flight path a bit, running the scenario again, noting the next crash, and repeating the process until the drone reached the end. Then the same was done for the remaining drones. The situation was complicated by the lack of any means for pinpointing flight path obstacles in the ARMA editor.

### Lack of vehicle control

#### Cameras

Besides the lack of a zoom function, one of the issues with the drone cameras was a lack of control over their default position. For experimental purposes, participants did not have access to camera controls outside of opening or closing a drone's feed. Unfortunately, unless the drone's camera was being actively controlled, it rested in a default forward-facing position. Ideally, we would have had it facing down, allowing participants to more easily spot people on the virtual ground. As it was, the forward position meant that the drones had to fly lower than would otherwise be required for people on the ground to be visible within the camera's viewing angle – which then put them at greater risk for tree encounters.

#### AI

The degree of AI guiding the drones could be adjusted using a slider. At higher levels, the drones would calculate optimum routes between assigned waypoints, and would also take care to avoid any hostile units it encountered – though they were not quite smart enough to avoid trees. Even at the lowest levels, though, the drones would sometimes take shortcuts between waypoints, most notably when they were turning a corner in a spiral flight path (see Figure 3). That made it difficult to precisely sequence the drones' encounters with targets, because the location of a given drone at a given time would vary from what had been calculated in the editor. As a result, multiple sessions of trial and error were necessary to get the timings right.

#### Unpredictable drone behavior

The drones had some behavioral idiosyncrasies we could not figure out and which hampered scenario creation. In particular, although they would obey any flight altitude we set that was below 10m, anything above that caused the drone to shoot up to 30 meters or more – far too high to easily spot and identify targets. The reasons behind this behavior are unclear, but they were particularly troublesome because an altitude of 11-12 meters would have been sufficient to avoid most of

the treetops while still keeping targets visible.

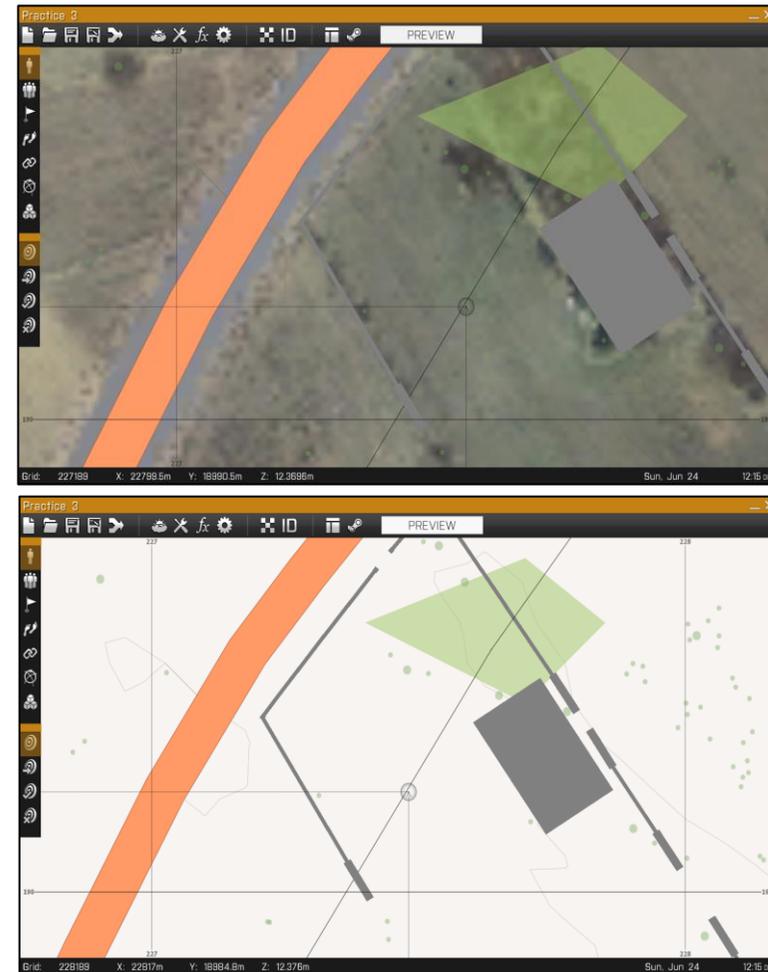
### Precision editing

The ARMA editor was great for game scenario creation in many respects, but it lacked some features necessary for the kind of precision generally sought in controlled experiments.

For example, as mentioned, we needed to sequence when the drones encountered each target so that (1) only one target was detected at a time and (2) there were short intervals between detections. The editor lacks a timeline tool for easy management of temporal events, meaning we had to sequence target encounters by calculating exactly when and where each drone would be at any given time. Those calculations depended on knowing how far a drone would have traveled at a given point. Unfortunately, the editor also provides no easy way to measure distance. The only distance reference tool available is a grid overlaying the map, with the distance between grid points equivalent to 10m, 100m, 1000m, etc., depending on zoom level. In order to use that grid for precise sequencing, we had to lay out the drone flight paths in straight lines along it.

More problematic were two issues that had no workaround. First, the editor map shows *some* of the obstacles that a drone might encounter, but not all of them. As a result, although a drone's flight path may seem to lead it through an empty plain according to the editor map, there may actually be a few trees in the virtual world that are sufficient to hinder the drone. And even when clumps of trees or bushes *are* visible on the map, the actual "drone danger area" around each such clump is unknown. Thus, routing the drones was a trial-and-error process that could have been so much easier.

Complicating that process was the second issue, which is the inability to determine exactly *where* on the editor map that a crash occurred. When a drone crashed, we had to use a combination of landmarks, waypoints reached, and estimated flight time to guess where the drone might have crashed and which was the offensive tree. Once located, simply adjusting the path slightly to the side was often an insufficient solution, because that could bring the drone into contact with other trees standing nearby but not apparent in the editor.



**Figure 7: An extremely zoomed-in view of the map editor is still insufficient for identifying potential tree-strikes. Only a subset of obstacles are actually shown.**

## RECOMMENDATIONS

The lessons learned during our attempt to use a virtual game world as a controlled environment in scientific studies may prove valuable both for game designers interested in expanding their market, and for other researchers considering the use of serious games in scientific studies. Hence, we have formulated a set of general recommendations for features that researchers and game designers alike should consider when choosing or building a game for such pursuits.

- *Adjustable realism:* The realism of the virtual environment should be configurable, allowing the game administrator to create both highly realistic scenarios and, when needed for control, unrealistic scenarios. For instance, one should be allowed to speed up or slow down time, adjust collision detection (e.g. make elements in the virtual environment intangible), and control the freedom of the players (e.g. where they can go, how they can interact).
- *Advanced scenario creation:* Functionality should be in place for the game administrator to create advanced scenarios that can progress and change over time. The administrator should have full freedom to control the presence, position, size, look, and behavior of game entities and the environment.
- *Real-time editing:* The game administrator should have the authority and tools to control conditions, characteristics, entities, and chain of events in the virtual environment during in-game sessions. This recommendation relates to how a facilitator may want to change experimental conditions of independent variables during an experiment.
- *Variable management:* Preferably, one should have in-game mechanisms for defining and logging variables such as the time spent to complete certain objectives, distance travelled, inflicted damage, and so on. This could simplify the data collection process and also reduce the time and resources required to run the experiment.
- *Configurable AI:* Artificial intelligence should be customizable to an extent that allows the game administrator to predict (with great precision)

how entities in the game will behave. This is essential for controlled experiments in which scenarios involve non-player entities. In addition, it should be possible to disable AI completely when necessary.

- *Controllable randomization:* Randomization of parameters (e.g. weather, entity positions) should be controllable so that the game administrator may reproduce certain random conditions as needed. Typically, this can be done by representing a specific randomization via a seed, such as a string, that users can then save and reuse to reproduce that randomization. This could simplify the process of recreating certain experimental setups.
- *Distributed simulation:* The game itself should be open in the sense that it (more easily) allows external software to access its data in real-time. This can, for instance, be achieved through the use of standardized architectures such as High Level Architecture<sup>4</sup> or Distributed Interactive Simulation<sup>5</sup>.
- *Recording and replay:* In-game mechanisms for recording and replaying previous tests/scenarios should be available to support data collection concerning player performance.
- *Timeline:* Complex, controlled scenarios often require a precise sequencing of events, such as timing of target detections across drones in our project. A timeline tool for visualizing and managing all the temporal events within a scenario would greatly simplify that process.

Several of the recommendations involve giving experimenters more control of the virtual environment, variables, and conditions that are being tested in an experimental setup, including the ability to remove all factors that may interfere with the results. If a game allows one to exercise such control, it could open up new possibilities that are otherwise impossible due to the amount of time and resources needed to create a realistic testing environment for crisis management in a virtual setting.

<sup>4</sup> <http://standards.ieee.org/findstds/standard/1516-2010.html>

<sup>5</sup> <http://standards.ieee.org/findstds/standard/1278.1-2012.html>

## REFERENCES

1. Connolly, T.M, Boyle, E.A., MacArthur, E., Hainey, T., & Boyle, J.M. (2012) A systematic literature review of empirical evidence on computer games and serious games, *Computers & Education*, 59, 2, 661-686.
2. Parasuraman, R., Cosenzo, K. A., & De Visser, E. (2009) Adaptive Automation for Human Supervision of Multiple Uninhabited Vehicles: Effects on Change Detection, Situation Awareness, and Mental Workload. *Military Psychology*, 21, 2, 270–297.