Military confrontations simulator for the training of army officers

TIAGO PEREIRA,[1] PEDRO A. SANTOS[2]

ABSTRACT

In this paper we present a study about the requirements of a military simulator for officer training, together with an architecture for their implementation in existing commercial frameworks which are low-priced or free, and which allow for the development of a constructive simulator. The proposed system has the advantage of being more affordable than existing military simulators. To demonstrate the viability of using one of the studied frameworks to develop a military simulator, a prototype was developed and tested with the target audience (military personnel). From its tests in can be concluded that the developed prototype and consequentially the created model, can fulfil the proposed objective.

KEYWORDS

Military simulation;
Low cost simulation;
Wargame;
Warfare

[1] Instituto Superior Técnico / INESC-ID, Universidade de Lisboa, Portugal
pedro.santos@tecnico.ulisboa.pt

[2] Instituto Superior Técnico / INESC-ID, Universidade de Lisboa, Portugal
tiago.f.pereira@tecnico.ulisboa.pt

1. INTRODUCTION

As modern military training can be very costly in terms of time and resources, it must be as efficient as possible. A part of that cost comes from operating military equipment, like weapons and vehicles, during training. A way to reduce that cost is to combine realistic simulation with the usual forms of military training. In this work, a realistic simulator is one that has a behaviour that is close enough to its real counterpart for training purposes. While simulation does not completely replace training with the actual equipment, as it cannot emulate all of its particularities, it can reduce the time soldiers need to use the actual equipment to learn how to use it, saving both resources and time, since a simulator is always ready, but the environment may not always be favourable for training with the actual equipment. For officer training, simulators allow for the construction of environments where the various entities and realities of warfare are simulated. As such, these simulators allow for officers to experience the stress and pressure of those kinds of situations, without employing the actual equipment or spending the necessary resources and space to simulate them in the real world in military exercises.

Realistic military simulators are complex and costly to buy and maintain. For example, VBS3, one of these military simulators, has a cost of $3,000[3] per seat, disregarding the price of the computers to run the software. The prices for other simulators in the market are not public. Furthermore, their use implies that the military officers in charge of training must be familiarized with the simulator to be able to design and create training sessions with it. This paper attempts to demonstrate that it is possible to create a military simulator, with the required characteristics to be successfully used in the training of officers, relying only on cheap or free frameworks, lowering the cost of that simulator.

[3] While this price was obtained from the Bohemia Interactive store, the web page is not directly available from their website. The page's address is: https://store.bisimulations.com/products/VBS3-Seat-License, accessed on 7th of May

Some questions emerge: Which are the requirements of a military simulator for officer training? Is it possible to use an open framework or a game engine to create a more affordable simulator? And will that simulator be able to deliver a sufficiently accurate situation to be used for the training of military commanders?

In the present work we will:

- Make explicit the main attributes which create a realistic military simulation;
- Define the requirements of a virtual simulator to train officers;
- Suggest a possible system developed using free or cheap development tools which can be used to simulate war situations to train military officers;
- Describe the results of a Proof of Concept (POC) that was developed to demonstrate the capabilities of the studied tools and of the proposed model.

In Section 2 some theoretical concepts essential for the understanding of this paper will be explained. In Section 3, the requirements for a military simulation are presented. Section 4 explains the model which answers those requirements while Section 5 describes how the proof of concept was implemented, tested and validated. Finally, the document's conclusions are presented.

2. THEORETICAL BACKGROUND

There are several concepts that describe a military body's structure, its composition and behaviour. For composition, the military uses Tables of Organization and Equipment (TOE), which describe military units with regards to their mission, capabilities and internal structure in terms of units and equipment (Headquarters Department of the Army 1997). For a unified behaviour, the military defines its military doctrine, which specifies a framework for the various actions to be performed. A military doctrine usually comes from the core beliefs of the military, standardizing the conducted operations and providing a common lexicon for the various leaders and planners to use in their communications (Jackson 2017).

The second concept which must be studied is simulation. Simulation exists in three different variants: live, virtual and constructive. Live simulation involves real people operating real systems. For example, when a tank is equipped with a live simulation system, it uses laser pointers to determine where a fired shell would land, actual shells thereby not being spent. Virtual simulation is described as real people operating virtual systems. An example of virtual simulation is the use of a simulator with specialized controls (reproduction of the vehicle's cockpit). Finally, constructive simulation

consists in having simulated people operate simulated systems. An example would be an entire virtual scenario where every entity is controlled by a computer, according to pre-defined rules. In a military context, constructive simulation is used as a Decision Support System (DSS) to determine the best approach to a given situation (Santos 2012).

Although military simulators for the training of military officers are wargames, they differ from videogames because of their purpose: as they seek to create a realistic simulation, the duration of the sessions, with longer ones and the characteristics of the units, in terms of both their equipment and behaviour (their interactions with the environment and the tactics employed), will be analogous with reality. To study how these concepts are actually employed in simulators in use by military forces, we analysed three different simulators:

- Tac Ops4;
- Masa Sword;
- MÄK Combat Staff Training.

It could be gathered that these simulators share some of their characteristics, which we will describe in the next Section and make explicit the requirements of a military simulator.

3. MILITARY SIMULATOR REQUIREMENTS

The following requirements contribute to understanding what is to be expected of a military simulator. We have drawn both from the preliminary analysis of the requirements of a constructive simulator proposed by (Cunha 2011), and from our own interviews with the officers responsible for constructive and virtual simulators, conducted at the Portuguese Military Academy[4] at their simulation centre, and at the Institute for Higher Military Studies.[5]

3.1 ARCHITECTURE REQUIREMENTS

Architecture requirements deal with the general aspects of the simulator, discussing its purpose and main components.

The system should be designed to help train any officer, offering the possibility of being used by the different scales of command, as

[4] Responsible for the training of the lower echelons of military command.

[5] Responsible for the training of the higher echelons of military command.

it should be possible to change the scenario dimensions and the size of formations. The sessions should run in real-time with the possibility of manipulating the time scale in order to suit the session's needs and purpose. This way, the trainee can experience the pressure of a war situation and understand how timing is relevant in military decision-making.

In order to reduce deployment costs, the system should be distributed, with a server making the calculations and the clients sending commands to the server (through orders) and visualizing the simulation. As the system is distributed, only the server will require a bigger investment, as the solution should be lightweight for the terminals.

When connecting to a session, the new client can play different roles, in accordance with the training officers' needs.

The different roles are:

- **Officer** – plays the role of a commander, controlling part of the simulated forces;
- **Instructor** – umpire role. It can influence the scenario status (changing the timescale or other aspects of the session), can issue orders to all the forces and introduce new units at any time during the session;
- **Radio Operator** – does the mediation between the trainees and the high command (Trainer), requesting air or artillery support or sanitary operations. In this role, the user will not see the simulation.

It should be possible to record each session, integrating the orders taken by each faction, the evolution of the state of the simulation and the communications in the different channels. The information can then be selected to preview and export to a video file.

3.2 UNITS REQUIREMENTS

Units requirements determine how the simulated units should be designed inside the simulator. Since the purpose was to create a military training tool, the behaviour and equipment was based on armed forces around the world. The behaviour is based on the military doctrine, but it should be customizable via the Standard Operating Procedures (SOP), for either a subset or all the controlled units. For example, if the commander desires to place scouts near the front, they should not engage enemy units.

During the interview with military officers of the Military Academy, it was mentioned that it should be possible to configure areas where a given unit should open fire if it sees an enemy unit.

Engineering units should allow for changing the terrain by building bridges or entrenching a position, helping friendly units or hindering enemy units by deploying minefields or other obstacles.

Other orders that were considered to be important, for the infantry units, are their ability to garrison in structures, upgrading the unit's line-of-sight and resistance, and the possibility of boarding vehicles.

3.3 MECHANICS REQUIREMENTS

Lastly, mechanics requirements discuss general functionalities that the simulator should contain.

The scenarios where the simulation session takes place should be possible to create and edit. They should take place in real locations on the world and so the simulator should be able to import terrain information in order to create them. The generated terrain is one of the crucial parts of the simulator as it influences the unit's speed (some units cannot move in all kinds of terrain) and line of sight.

Another important system is the weather system as it affects the scenario as a whole, changing the unit's line-of-sight and movement capabilities. As with all other features, clients connected as an instructor can change the weather at any given time. Other features that should exist in the simulator are:

- Sanitary and logistic operations;
- Artillery and air missions;
- Malfunctions;
- Information operations;

Regarding sanitary and logistic operations, they should be represented from their inception until their end, there being the possibility of being disrupted by the enemy. This contributes to provide a realistic simulation as in the battlefield any units are subject to enemy fire.

Concerning the artillery, fire missions should distinguish between planned fire or non-planned fire, affecting the time that the artillery needs to fire and reflecting the calculations the artillery crew needs to make before firing. Besides the type of fire, it should also be possible to choose the munitions and the number of salvos of each fire mission. Air missions are to be configured in a similar manner.

Finally, regarding malfunctions, they could happen at either a unit level (vehicle malfunction, weapon jamming), which can disable or reduce the efficiency of the units or at the communication level.

Malfunctions can originate from either electronic warfare or from sabotage. Other features found to be required are:

- Stacking of the unit markers when they are near each other on the map, in order to reduce clutter. The various units contained in the stack are then accessed via the context menu when the stack is clicked.
- The generated map should display a grid over it, like military maps, so that trainees must calculate the positions to reference them. Instructors, however, can access the position calculations directly and can draw over the map, if needed.

The coming section presents our proposed architecture to satisfy the requirements identified.

4. SOLUTION ARCHITECTURE

This Section describes our proposed solution for a realistic military simulator. The requirements identified in the preceding section allow the system to provide realistic behaviour.

4.1 SOLUTION MODULES

Since the proposed system is complex, it was divided in different modules (as seen in Figure 5):

- **Simulation Interface** – this module will contain the code required to draw the interface through which the client interacts with the simulation. The interface changes depending on the user's role.
- **Simulation Database** – the system will keep the various types of information used by the simulator (Table of Organization and Equipment, Doctrines, Formations, military equipment, maps, scenarios) in a database;
- **Scenario Database** – module to be used by the trainers to create and store the scenarios for their trainees by combining the various types of information in the simulator's database;
- **Simulation server** – central module of the system which will do most calculations required by the simulation, like hit calculation and movement processing;
- **Voice Communications** – Voice-over-IP (VOIP) module which will guarantee voice chat between the different instances of the simulator, with the option of choosing between channels;
- **HLA/DIS Interface** – this module will implement the HLA and the DIS standards in order for the projected simulator to communicate with other compliant simulators;
- **Archives** – will offer the possibility of recording the played scenarios for after-action reviews, further enhancing the learning possibilities. The recording will display the orders given by the

different factions throughout the duration of the play, the communications between the different players allowing to see the action developing on the map.

They are connected in the manner shown in the following scheme:

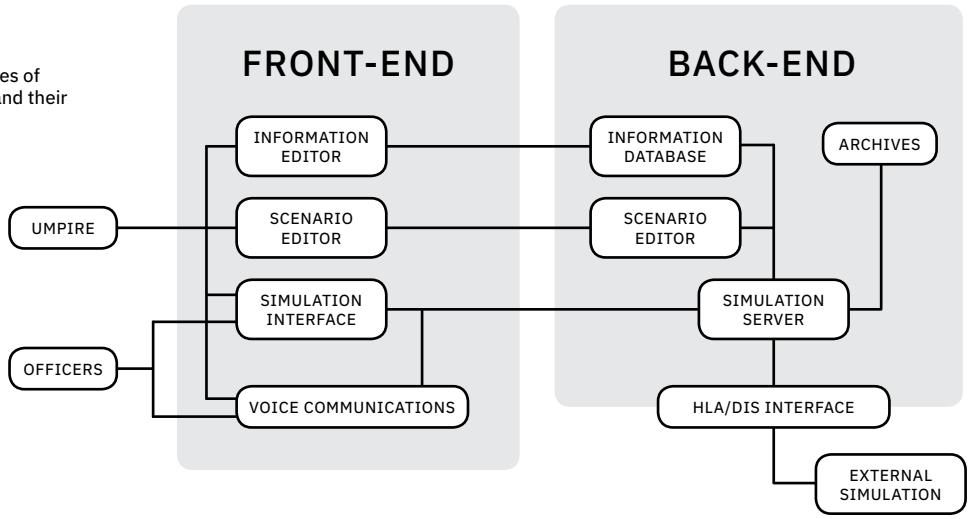


Fig. 1
Internal modules of the Simulator and their relationships.

The units will be kept in the information database and characterized by different attributes, depending on their type, for instance:

- Equipment used by the unit;
- Ammunition – kinds and quantities of ammunition currently in possession of the unit;
- Armor (divided into front, back, sides and top), when applicable;
- Movement capabilities in the different kinds of terrains.

In addition to these characteristics, the unit's experience, status (under fire, moving, standing still or others) as well as the terrain, both the one at the unit's location and the one crossed during the bullets voyage, will influence the capacity of the unit to engage effectively the enemy units.

The units are organized hierarchically by using an implementation of a TOE. More specifically, a TOE's internal structure is implemented using two types of nodes: Basic nodes and Composite nodes. Both nodes share the attributes derived from their representation during the simulation like their NATO symbol, size symbol and its type (combat unit, support unit). However, some attributes, like the unit's equipment or its soldier count, depend on the subunits that it is made of. More specifically, in a basic node, it is possible

to define directly the number of soldiers that the unit contains and the equipment it uses, as well as the attributes mentioned before. When creating composite units, it is not possible to define all the same attributes as in the basic units: their values will be automatically known by considering the subunits that are added to them. These subunits can be either basic nodes or other composite nodes.

As such, the person that is creating/ editing a TOE starts by defining the equipment that a unit can use, then the basic units and then the composite units using those basic units and other composite nodes, creating a tree like in a real TOE.

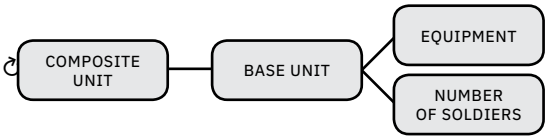
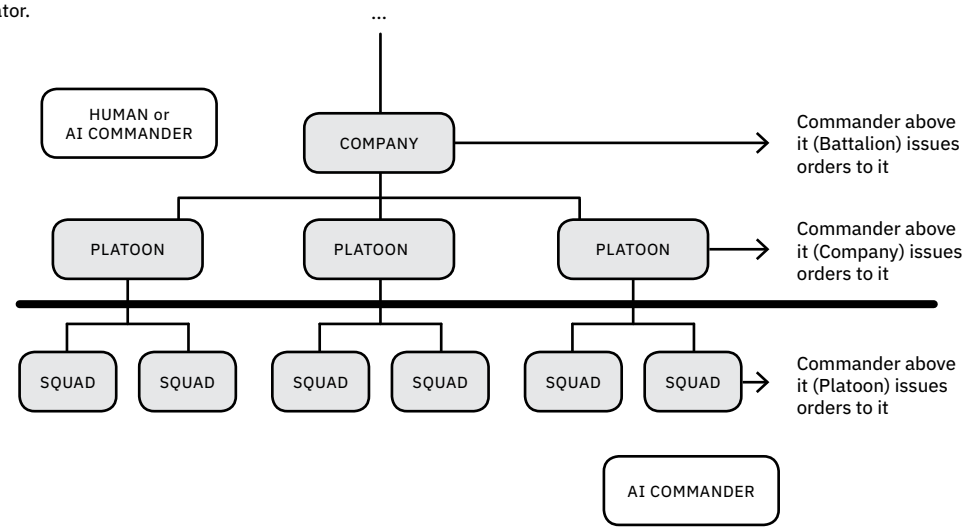


Fig. 2
Internal constitution of a TOE instance.

The internal components of a TOE can be seen in Figure 2. The TOE is used to define two other types of information: Formations and Doctrines. While TOE usually defines an abstract unit, Formations are used to define specific units, using as a base a specific TOE. For example, if we wanted to define an infantry company called the 4th Infantry Company, first we would have to define what is an infantry company, and then, by using that TOE as a base, we would construct a Formation with the desired name. A Formation allows us to create instances based on the units created in the TOE and so customize their characteristics, like their name, to create a unique unit. Formations are the entities that will be spawned and controlled by the commanders during a simulation and whose subunits are assigned orders by their commanders. For example, a company commander assigned to 1st Company as defined in Figure 3, will issue orders to all its direct subunits, the three platoons.

Fig. 3
Hierarchy and commanders in the simulator.



Doctrines are used to define the orders which are executed by each unit, as defined in a TOE. As such, the orders are hierarchical by nature. When a unit, controlled by an Artificial Intelligence (AI), receives an order, its doctrine will determine how it will be translated into orders that can be executed by the unit.

These orders are constructed using actions, which can be of three different types: basic, composite or general. Each action within an order is associated with a specific subunit.

Actions can be associated with an interrupt condition, which allows for them to have another way to be completed, allowing a greater control of the timing in which the subunits execute their orders. For example, if a movement order is issued to a given point but it has a condition of being near another determined unit, then this order will end either when the unit has reached that destination or if it is close to that pre-determined unit.

General actions are the ones which can be added to any level of the hierarchy. An example of such an action is the wait action. Basic actions are implemented on the units themselves, as they correspond to the actions that any trained soldier can execute and are used to construct the composite actions. They are associated to any echelon which is not associated with an officer, like squads or fireteams. Both general and basic actions are static in the sense that a user cannot add new actions of these types, only the developer by editing the simulator internal code.

Finally, composite actions are the ones which give the simulator flexibility by providing the users with the ability to construct any manner of orders, by applying together the various action types. Composite actions are built in any echelon which does not have basic actions and each one corresponds to an order that can be given to the unit during a simulation. When a user defines a new order, he can construct different action sets, which allows for the same order to be completed by the AI in different ways, depending on the conditions associated with each one. If there is more than one condition which is verified at a time, the action set is chosen from the restrictiveness of that condition. For example, if an order for movement was created for a platoon but the manner in which that platoon moves is dependent on expected danger during the order's fulfilment,[6] then different action sets would be created, each one with an appropriate condition, which would allow for the AI to have a behaviour similar to that of a platoon if it had been commanded by a human.

A doctrine uses the logic of a Hierarchical Task Network (HTN) (Russel e Norvig 2003) decomposer to construct a hierarchy of the orders:

1. Orders given to units above squad level are all composite orders (or composite tasks);
2. The doctrine describes how a composite order is decomposed into a set of orders belonging to the level below (either themselves composite or basic actions);
3. The decomposition ends when the given order has been transformed into a set of basic actions;
4. Each order can have attached to it constraints, which finish the order prematurely.

TOE, Doctrines and Formations, are related in the manner depicted in Figure 4.

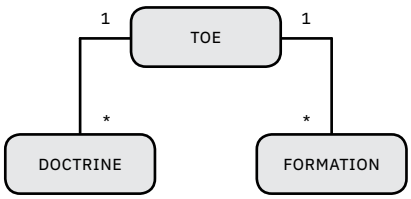


Fig. 4
Relations between the concepts of TOE, Doctrines and Formations

[6] The video "The Rifle Platoon Dismounted Movement Techniques", available on https://www.youtube.com/watch?v=-qdFd9Uh0N0, for instance, shows how an infantry platoon adapts their movement tactics depending on the contact probability.

Also contained in the database are the maps used to create the scenarios. These are loaded and interpreted by the simulation that defines 2D representations of the terrain (with a distinction between types of terrain and height) as the setting for the simulation sessions.

In addition to having the stacking behaviour described earlier, the units can be merged (if compatible) or separated as necessary. When there is a stack of independent units, there will be a visual cue informing of the situation.

As the proposed architecture is highly complex, a POC was developed. The following Section describes the POC that was developed to demonstrate the feasibility of creating a military simulator by using a cheap or free development framework and how that solution achieves the purposed objectives.

5. SOLUTION IMPLEMENTATION AND VALIDATION

This section describes the POC that was implemented to demonstrate the feasibility of the model described in the previous section. We will also describe how we tested our POC, how we prepared the tests, how they were conducted and their respective results. As the POC was created to show the potential that our proposal has, our tests were aimed to demonstrate that the current internal calculations of the simulation could be deemed as realistic.

Having analysed three different options to create a military simulator, what was chosen was *Unreal Engine*, justified by these facts:

- Using *Unreal Engine* allows for just focusing on the actual logic of the simulator instead of how to code it, because of the existence of blueprints. Without them, it would be necessary to reason what exactly would be the correct library to use or if it was necessary to code it.
- *Unreal*'s community is active, which eases the process of understanding the reason for any difficulty that emerges during development;
- There is a large quantity of *Unreal Engine* tutorials online;
- *Unreal Engine*'s documentation is extensive;
- Nodes were created by experts in both *Unreal Engine* and *C++*, which guarantees a certain level of performance when using them;
- *Unreal Engine* also has a marketplace where there are free plugins which extend the functionality of the engine;
- The client-server communications in *Unreal Engine* are close to the simulator's goal, where the server does all the calculations and the client shows the results from those orders.

Although some of these features are also present in other game engines, like *Unity*, we have chosen *Unreal Engine* due to previous experience in working with it.

This POC's objective is to implement a simplified version of the proposed architecture to demonstrate its potential to create a lower priced military simulator and the possibilities that a game engine such as *Unreal Engine* offers for this kind of project. The developed POC focused in implementing features of three of the modules discussed in Section 4.1, namely the Simulation Interface, the Simulation Server and the Unit Database.

5.1 SIMULATION SERVER IMPLEMENTATION

The Simulation Server was constructed by making sure that functions which have an impact on the gameplay are called in the Server version of the various entities in the game world, and so its code is spread across the various classes.

5.2 DOCTRINE

As stated in Section 6.1, the doctrine will be used to customize the behaviour of the AI controlled units, and so, separate the unit's implementation from its behaviour in the simulation.

In the POC, this was implemented by separating the functionality of a doctrine in separate classes, each one with a separate function within the overall functionality of our concept of a doctrine. The various classes are used sequentially during the decomposition of the orders. During the decomposition process, if there is an interrupt condition attached to the action, it is added to unit's blackboard, and is verified via a service in the behaviour tree, until either the condition is verified itself and the order is skipped, or the order reaches its supposed end.

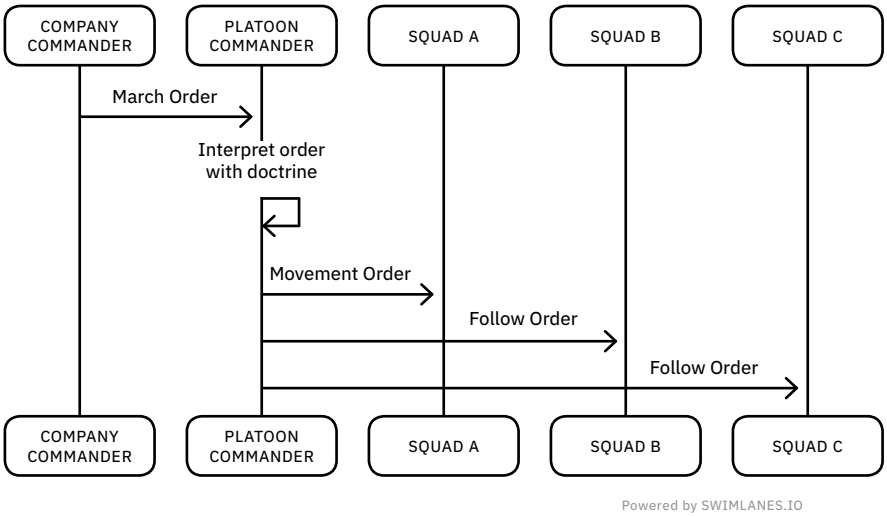


Fig. 5
Invocation of a composite order.

In Figure 5, it is possible to see an example of an invocation of a composite order. The commanders can either be human or an AI. If it is an AI-controlled commander, the doctrine will determine how the order is decomposed to be interpreted by the next echelon. If not, then the human commander must know how to decompose that order.

5.3 VALIDATION

In order to validate our prototype, we performed two sets of tests: one for the system usability, where we used the SUS (Brooke 2013), and another to evaluate its realism.

The first set of tests had the following results, scored from 1 to 100:

- 55 for the Information Editor;
- 56 for the Scenario Editor;
- 71 for the Simulation Editor;

These results point to serious problems with the system's usability, particularly on the Information Editor and the Scenario Editor. However, as the users who performed the tests were not the end-users for this system, these tests' results are not conclusive.

The second set of tests had the purpose of evaluating the realism that the current POC offers in terms of military behaviour and as such could only be performed by persons which had some degree of understanding of military tactics. More specifically, the tests were made by connecting 3 users to the simulator and having them play different roles, with 2 players occupying the roles of commanders and one occupying the position of the Umpire, alike the training done at the Military Academy.

We invited officers from the Military Academy to view the model and the developed prototype, to understand its state and receive feedback about its features.

After finishing the tests and the various officers having taken part in the exercise and demonstrations, the officers focused on the following points:

- The program is simple to understand and manipulate, reinforcing that understanding of military concepts is important to understand the program's interface and the flow of the program. Therefore, the interface and usability were considered adequate by the end-users.
- The program can integrate with other tools in a seamless way. For example, its ability to use real terrain information and having that terrain information directly translated into the simulated world. One of the major disadvantages of the current systems used by the military is the lack of interoperability between the various tools that exist. For terrain information, the military has a tool which allows to export all the relevant information about the terrain, such as, where does the terrain give cover and where do the different units are able to navigate.
- Despite being very basic in the POC, the unit's AI behaviour follows a doctrine, which is very important in a simulator.

However, the officers also pointed out some errors in our model and POC:

- Usually, the Platoon commanders do not micro-manage the infantry squads as was implemented. They manage the support squads (like those with mortars or machine guns) directly, but they command the infantry squads in a more general way. For example, it should be possible to define itineraries for those squads to follow instead of ordering them directly.

In the next Section, we present the conclusions of the developed work.

6. CONCLUSIONS

In this paper, a proposal for an architecture of a realistic military constructive simulator was presented. However, given the high quantity of requirements defined and the relative short time for development, only a proof of concept of this system could be constructed, in order to demonstrate the possibilities of low-cost development software to create traditionally costly software. To construct the proof of concept, *Unreal Engine* was used, as it allows for the rapid development of prototypes via the blueprint system.

The major contributions of this work, in terms of the requirements described in Section 4, are related to the architecture and units'

requirements: by creating units through TOE as described in Section 4, it is possible for the units to belong to any echelon and thus different scales of command can be simulated. The adaptable behaviour that the units should have is supported with a doctrine as described in Section 4. More specifically, this implementation allows for the creation of orders for any unit (from platoon level to the upper echelons) by defining orders for a given unit, using the orders defined for the units it is composed of. For example, if an infantry platoon is constructed from three infantry squads, the orders for the infantry platoon will be defined according to the orders available to the infantry squads. Besides the actions associated with each unit, the system chooses the most appropriate actions according to conditions associated with each set of actions.

A game engine, like *Unreal Engine*, offers the server-client architecture that is required. Regarding the mechanics requirements, by using a game engine, we are offered the AI functionalities required to differentiate the types of terrain and to create an influence map. Although *Unreal Engine* offers a great set of built-in features, these are skewed for a certain type of games. It was therefore necessary to work against the original purpose of the engine's features in order to repurpose them to a different type of creations, like the one discussed in this paper.

Note that by having the simulation engine built, new scenarios are relatively easy and fast to create, as all the previously used units can be repurposed for the new scenario. One just needs to import the physical terrain into the simulator and place the appropriate units to create a new training situation.

The programming work to produce this POC was 5 man-months. The results obtained from the tests demonstrate that our hypothesis (that it is possible to develop a realistic military simulator for the training of army officers using a cheap or free development framework) can be fulfilled by the model which we created. Furthermore, it can also be conjectured that if the proposed system is executed without assigning human players any position, the simulation can run by itself, transforming an otherwise virtual simulator into a constructive simulator.

6.1 FUTURE WORK

While it was possible to create a POC which demonstrates the potential of the proposed model, future work in implementing it should be done in code, as only then will it be possible to access the total potential of *Unreal Engine*. By using code, the simulator will become more efficient, more portable as well as more stable. As stated above, we focused mainly on the architecture and the unit's re-

quirements, as they were more important for our purposes, further requirements being left for future work. An engine like *Unity* would have had a better support for these types of programs, as users usually have to implement their own versions of features which are offered by *Unreal Engine*, making *Unity* more flexible.

We predict that to produce a first deployable version one would need about 18 man-months for the simulation implementation and 6 man-months dedicated solely to the interface.

AKNOWLEDGMENTS

We would like to thank all the army officers who collaborated with us during the development of this work. We also acknowledge the support of the Unreal Engine's community, particularly Victor Burgos, for helping with understanding the system's inner workings. This work has been partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC /50021/2013;

REFERENCES

Brooke, John. 2013. "SUS: A Retrospective." *Journal of usability studies* 8 (2): 29-40. Accessed 05 08, 2018. https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/.

Cunha, André. 2011. *O emprego do sistema de simulação constructiva como ferramenta de apoio à decisão; uma proposta ao exército brasileiro*. Master Thesis, São Paulo: Escola de comando e estado maior do exército.

Global Security. n.d. *Global Security*. https://www.globalsecurity.org/military/library/policy/army/toe/toenum.htm.

Headquarters Department of the Army. 1997. "Force Development and Documentation - Consolidated Policies." Washington, DC.

Jackson, Aaron P. 2017. "The Nature Of Military Doctrine: A Decade of Study in 1500 Words." *The Bridge*. 15 November. Accessed April 7, 2018. https://www.realcleardefense.com/articles/2017/11/15/the_nature_of_military_doctrine_a_decade_of_study_in_1500_words_112638.html.

Russel, Stuart J., and Peter Norvig. 2003. *Artifical Intelligence: A Modern Approach*. Person Education.

Santos, João. 2012. *A Simulação. Contributos para a formação e treino.* Master Thesis, Lisboa: Academia Militar. Direção de Ensino.

U.S. Department of Health & Human Services. n.d. "System Usability Scale (SUS)." *usability.gov*. Accessed May 08, 2018. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

Wayman, Erin. 2012. "What is War Good For? Ask a Chimpanzee." *Slate*. Outubro. Accessed December 6, 2016. http://www.slate.com/articles/health_and_science/human_evolution/2012/10/chimpanzee_wars_can_primate_aggression_teach_us_about_human_aggression.html.