Military Confrontations simulator for the training of army officers

Extended Abstract

Tiago Pereira
Instituto Superior Técnico
Lisbon, Portugal
tiago.f.pereira@tecnico.ulisboa.pt

ABSTRACT

In this thesis, a study about the requirements of a military simulator for officer training is presented. In order to understand what those requirements are, we had interviews with military officers responsible for officer training and analyzed wargames existing in the market, including tabletop games, video games and military simulators in use by military forces.

The study results in a comparison between the different military simulators in the market and a set of requirements for a military simulator. We also studied existing commercial frameworks which are low-priced or free, and allow for the development of a constructive simulator.

Combining both analyses, a proposal for a constructive simulator is given. 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. From its tests in can be concluded that the developed model can fulfill the proposed objective.

KEYWORDS

military simulation; low cost simulation; wargame;warfare

ACM Reference Format:
Tiago Pereira. Military Confrontations simulator for the training of army officers: Extended Abstract. In . ACM, Lisbon, LX, PT

1 INTRODUCTION

Warfare is an activity that is characteristic of human nature, as it can be concluded by analysing the history of Humankind history. Furthermore, ethnographic studies suggest that aggressive behaviour is part of the nature of primates of which humans are an example [7].

Throughout this long practice of warfare, better weapon systems were developed and given their growing complexity and lethality, armies around the world must be trained to use them effectively. The soldiers [1] must learn to operate the weapons systems and their commanders must learn both to adapt to the modern battlefield and how those systems are better applied.

As the necessary 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. By realistic simulator it is meant a simulator that has such a behavior that is close enough to its real counterpart for training purposes. While simulation does not completely replace training with the actual equipment, as it can not emulate all of its particularities, it can reduce the time soldiers need to use the actual equipment in order to learn to use it, saving both resources and time, since a simulator is always ready but the environment may not always be favorable for training with the actual equipment. In terms of 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.

2 MOTIVATION

From the previous introduction, it can be concluded that realistic military simulators are complex and hence, costly to buy and maintain. For example, VBS3, one of these military simulators, has a cost of $3,000 [2] per seat, disregarding the price of the computers to run the software. Furthermore, their use implies that military officers in charge of training must be familiarized with the simulator to be able to design and create training sessions with it. This thesis seeks to demonstrate that it is possible to create a military simulator, with the required characteristics to be successfully used in the training of officers, relaying only on cheap or free frameworks, lowering the cost of that simulator.

Therefore, 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?

3 OBJECTIVES

The present work will:

- Make explicit the main attributes which create a realistic military simulation;

[1] A soldier is distinguished from a commander in the sense that the first is engaging the enemy and the second is coordinating the movement of the first.

[2] While this price was obtained from the Bohemia Interactive store, the web page is not available directly from their website. The page's address is: https://store.bisimulations.com/products/VBS3-Seat-License, accessed on 7th of May, 2018

- Define the requirements of a virtual simulator to train officers;
- Suggest a possible system developed by using free or cheap development tools but which can be used to simulate war situations to train officers;
- Develop and test a Proof of Concept (POC) to demonstrate the capabilities of the studied development tools and of the proposed model;

The coming Sections are organized in the following way: in the next section some theoretical concepts essential for the understanding of this document will be explained. After it, the requirements for a military simulation are presented and then, the model which answers those requirements is introduced. In Section 8 the tests that were are presented, as well as their respective results. After that section, the document's conclusions are presented.

4 THEORETICAL BACKGROUND

This section's purpose is to explain the various concepts which are important to understand the model that was developed in the context of this document. As our purpose is to present a proposal for the creation of a simulation program for the training of armed forces, developed through cheap development tools, it is necessary to explain various military concepts and the existing simulation variants.Through our analysis of the wargames available on the market and their contributions to create a realistic simulation, we present our identified requirements for a military simulator.

The studied **military concepts** are the ones which give a military body structure its composition and behavior. For composition, the military uses Tables of Organization and Equipment, which describe military units in terms if their mission, their capabilities and their internal structure in terms of units and equipment [1]. For a unified behavior, the military defines its military doctrine, which specifies a framework for the various actions that the military performs. A military doctrine usually comes from the core beliefs of the military, standardizing the conducted operations and providing a common lexicon for the various military leaders and planners [3].

The second concept which must be studied is simulation. Simulation exists in three different variants: **live**, **virtual** and **constructive**. **Live simulation** is when there are real people operating real systems. For example, if a tank is equipped with a laser pointer so that it does not fire the usual shell but it is possible to know where that shell would land if fired. **Virtual simulation** is described as real people operating virtual systems. An example of virtual simulation is when instead of using an actual vehicle to train its use, a simulator is used. Finally, **constructive simulation** is when simulated people operate simulated systems. When a whole virtual scenario is created where every entity is controlled by the computer, according to pre-defined rules for those entites, it is an example of constructive simulation. In a military context, constructive simulation is used as a Decision Support System (DSS) to determine the best approach to a given situation [4].

Lastly, **wargames** exist in three variants on the market: **tabletop, videogames** and **military simulators**. **Tabletop wargames** are the oldest variant and thus can no longer be expected to deliver an accurate simulation. However, from the tabletop wargames came the more recent ones and it is therefore important to study them in order to understand what is essential in creating a realistic military simulation. The most important conclusions drawn from the analysis of tabletop wargames are:

- Air units are based on a certain airbase, which imposes limits to the quantity of air units in each one and their respective range;
- When there are too many units at one space, their movement is limited (costlier) and their combat effectiveness is reduced(stacking);
- Units establish zones of control around them, which represent the threat posed by the unit to enemy forces. This is particularly relevant for both turn based games and real-time games, as it influences the unit's behaviour during the game.
- The terrain is differentiated, which means that in different territories, the player will have a different movement capabilities or actions specific for that type of territory;
- Certain units are inherently strong against others, with the concrete example of tanks' armor giving them an advantage when fighting regular soldiers in open terrain;
- Weather greatly hinders the capabilities of the air units;
- In order for the units to be supplied there must be a clear line between the supply point and the unit, and it must be in a certain radius of the supply point;
- Units which are cut-off (not supplied) from the rest of its faction's units will lose morale, which will influence its offensive and defensive capabilities.

It is important to mention that while tabletop wargames are no longer used as a DSS, sand tables (an example of a tabletop wargame) are used to visualize the operations when planning them. For example, the Iraqi military used them to plan for the defense of Kuwait City. **Video games** were analyzed because as a computer is now being used to do the calculations, it is possible to create a more complex simulation, integrating what the tabletop wargames did and adding other details. The videogame analyzed, *Wargame: Red Dragon*, provided an accurate war simulation which outweighs all the other reviewed videogames. In this game, an accurate simulation is constructed by:

(1) Having a large quantity of highly detailed units (1200 units);
(2) The units having the capacity of using advanced tactics;
(3) Showing the mutability of the battlefield;
(4) The fog-of-war depending on the line-of-sight of the friendly units;

However, since it is a video-game, and as such it has the objective of being ludic and not realistic, there are some details which can not be viewed as realistic:

- The damage/repair model that the game implements is not realistic;
- The infantry units move too fast;
- Vehicle fuel depletes too fast;
- Infantry can engage and easily win against a tank in open terrain, which is not realistic;
- Reinforcement is almost instantaneous;
- The weather is not simulated.
- Unit behavior is not changed by changes in the environment (there is no unit morale);

These limitations are understandable given the duration of the play sessions (30 minutes to 1 hour) and the objective of the system. The purpose is exactly what distinguishes **military simulators** from video-games. These simulators are used as either a DSS or as a training tool and thus, the duration of the sessions, the characteristics of the units and their behavior are influenced. Three different simulators were analyzed:

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

From these analyzes it can be concluded that these simulators share some of their characteristics, which are included in the next Section and make explicit the requirements of a military simulator.

5 MILITARY SIMULATOR REQUIREMENTS

The following requirements were obtained from the analysis of a paper, in which a preliminary analysis of the requirements for a constructive simulator was made [2], and from interviews conducted both in the Portuguese Military Academy [3], at their simulation center, and at the Institute of Superior Military Studies [4], with the officers responsible for a constructive simulator.

Through the following requirements, it will be better understood what is expected of a military simulator. The requirements are numerous and so they have been divided in three categories:

- Architecture;
- Units;
- Mechanics.

5.1 Architecture Requirements

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

The system should be designed to help train any officer, offering the possibility of being used by the different scales of command, as it is possible to change the scenario dimensions and the size of the formations used.

The sessions should run in real-time with the possibility of manipulating the time scale in order to suit the session's need and purpose. This way, the trainee can experience the pressure of 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:

(1) Officer - plays the role of a commander, controlling part of the simulated forces;
(2) 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;
(3) 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. It should be possible to choose what is presented, and export it to a video file.

5.2 Units Requirements

These requirements determine how the simulated units should be designed inside the simulator. As the purpose of this thesis is to create a military training tool, the behavior and equipment should be based in armed forces around the world. The behavior is based on the military doctrine, but it should be customizable via the Standard Operating Procedures (SOP), for either a subset or for 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 units.

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, changing, for example, their movement speed.

5.3 Mechanics

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

The scenarios where the simulation session take 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 such scenarios. 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 system which is important is the weather system as it affects the scenario as a whole, changing the unit's line-of-sight or movement capabilities. As all the simulation facets, 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, existing 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.

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

Concerning the artillery, fire missions should distinguish between planned fire or non-planned fire. The distinction affects the time that the artillery needs to fire, reflecting the calculations the artillery crew needs to make before firing. If before beginning the session the trainees requested that position would be a possible target for artillery fire, then the fire mission will execute as planned fire. Otherwise, it will be a non-planned fire. 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 that are required for a simulator for the training of officers were:

- Stacking of the unit markers when they are near each other on the map, in order to reduce clutter. The various units which are 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 on 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.

6 SOLUTION ARQUITECTURE

This Section describes our proposed solution in order to develop a realistic military simulator. This solution will take into account the military simulator's requirements described in the previous Section, which will allow for the proposed system to provide realistic behavior, in the context of this thesis.

6.1 Solution Modules

Since the proposed system has a certain degree of complexity, it was divided into different modules (as seen in Fig. 5):

- Simulation Interface - this module will contain the code required to draw the interface through which the client interacts with the simulation, which changes depending on the user's role.
- Scenario Database - module which will be used by the trainers to create the scenarios for their trainees by combining the various types of information in the database;
- Simulation Database - the system will keep the various types of informations used by the simulator (Table of Organization and Equipment (TOE), Doctrines, Formations, military equipment, maps, scenarios) in a 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 in the map.

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

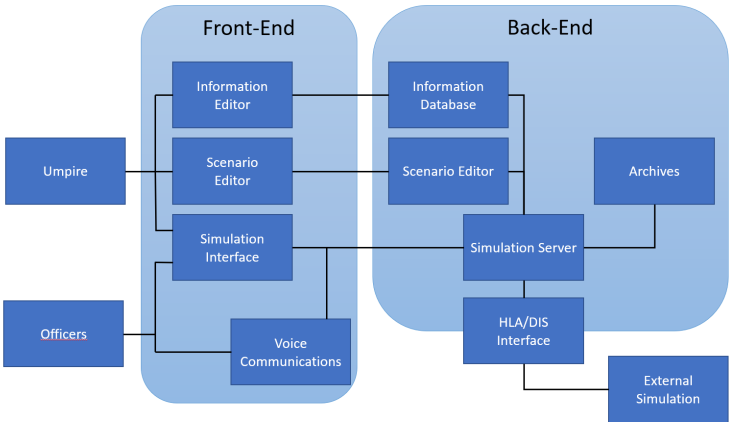

Figure 1: Internal modules of the Simulator and their relationships.

The units will be kept in the information database and characterized by different attributes, depending on the type of unit:

- 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;

During the preparation or execution of a training scenario, there will be more attributes:

- Health/Condition - state of the unit. For vehicles, the condition will reflect what the current malfunctions (locomotion or weapons) of the vehicle are, and for human troops, the health attribute will show the status of the units within a squad;
- Morale - affects the fighting capability of the unit and is influenced by the supplies the unit has, the suppression it is receiving and its health. After a certain threshold, the units will try to retreat, disobeying the commander's orders;
- Fuel (when applicable).

The weapons themselves also have characteristics such as:

- Name;
- Caliber;
- Range (divided into practical, useful and effective);
- Type of target they engage, between Unarmored, Armored or Air targets;
- Firing rate;

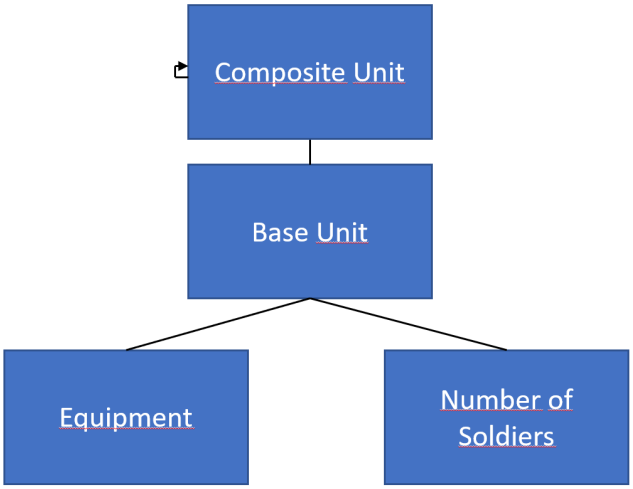

Figure 2: Internal constitution of a TOE instance

- Suppression - degree to which they affect the morale of enemy units when used against them;

As well as 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 taking into account 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 uses, then the basic units and then creating composite units using those basic units and other composite nodes, creating a tree like in a real TOE.

The manner in which a TOE is build internally can be seen in the figure 2.

The defined TOE are used to define two other types of information: **Formations** and **Doctrines**. As a 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 a infantry company called the 4th Infantry Company, first it would have to be defined what is an infantry company. Then, by using that TOE as a base we would construct a **Formation**. A **Formation** allows us to create instances based on the units created in the TOE and then customizing 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.

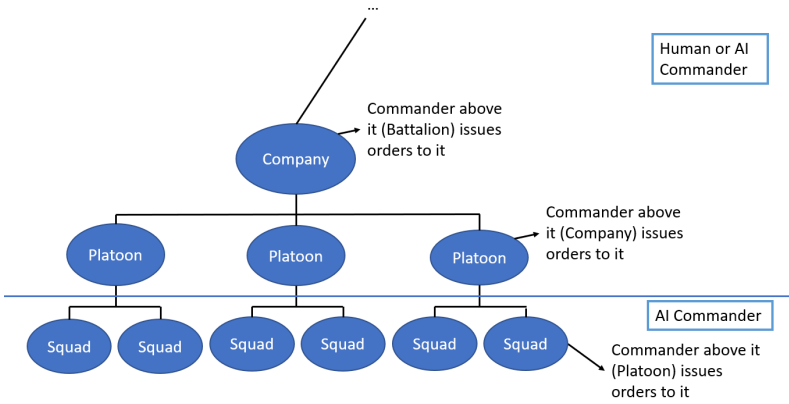

Figure 3: Hierarchy and commanders in the simulator

Doctrines are used to define the orders which a unit, existing in the its associated TOE, can execute during a simulation. When a unit is controlled by an Artificial Intelligence (AI), the doctrine will determine how the decomposition of orders is executed, by defining the various orders that can be given during the simulation.

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

All actions have associated with them an interrupt condition, which allows for them to have a condition which also result in their completion, allowing a greater customization of the timing in which the subunits execute their orders. For example, if a movement order is issued 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. They are orders which any unit can receive regardless of their echelon. 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 it with the ability to construct any manner of orders which the user would need to design, by grouping together the various action types. Composite actions are build in any echelon which does not have basic actions. Associated with

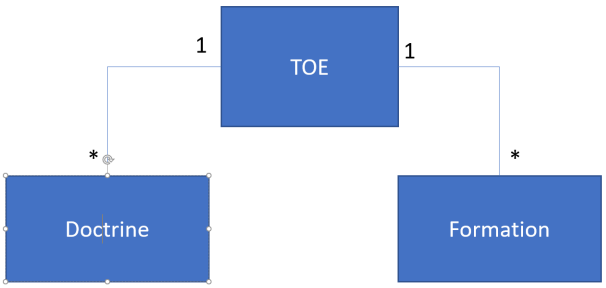

Figure 4: Relations between the concepts of TOE, Doctrines and Formations

composite actions is also the concept of action sets. They allow for the same order to be fulfilled in different forms by the AI depending on the conditions associated with the action sets. 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 at the final destination [5], then different action sets would be created, each one with an appropriate condition, which would allow for the AI to have a behavior similar to that of a platoon if it had been commanded by a human.

A doctrine uses the logic of an HTN 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 task is decomposed into a set of simpler actions (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.

Apart from the orders themselves there is another element which is translated in each step of the its decomposition: the order's parameters. Parameters are defined for each order and vary in terms of their:

(1) Type - Indicates what the parameter's value is. For example it can be a distance or a reference to another unit in the formation;
(2) Multiplicity - Indicates how many values a parameter has associated with it;

A parameter's value can come from the order above or can be explicitly defined, being for example a reference to another subunit at the current level or a numerical value. During the decomposition, the initial parameters (if there are any) are mapped to the parameters of the orders below according to what was defined in the doctrine.

[5]The video "The Rifle Platoon Dismounted Movement Techniques ", available on https://www.youtube.com/watch?v=-qdFd9Uh0N0, for instance, shows the manner in which an infantry platoon moves varies depending with the contact probability.

Having described these three artifacts, TOE, Doctrines and Formations, it is shown that they are related in the manner pictured in 4.

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.

Through a scenario editor, it is possible to prepare training sessions using the simulator, created from all the other artifacts which exist on the simulator's database.

A training session is made from both a location and from factions which exist in the location, for the purpose of the session.

A location is chosen from the list of the available maps. The factions are constructed by choosing its name, the TOE it uses, the doctrine it follows (associated with the chosen TOE) and the formations which belong to that faction (associated with the chosen TOE).

As each formation is unique, as it represents a specific unit, it can only be added to a single faction. After a formation is added to a faction, it is possible to change the position of each of the units that compose the formation and to define their initial orders, (whose kinds depend on the faction's doctrine) to be given when the session starts, and to define generic objectives which will guide both the AI and human commanders by to giving them a context to operate.

Those initial orders are carried out in a different way depending on whether the unit is controlled by a human or by an AI: in the first case, the order is not immediately carried out but instead given in the form of a objective displayed on the interface. In the second case, the order is carried out in accordance with what was defined in the doctrine as the procedure to execute that order. If an order is given during a session, the same happens.

After the scenario is defined in a satisfactory way, it can be used to create training sessions where either a human commander or an AI is assigned to command a formation.

In the simulator itself, the units will be given orders via the following procedure:

(1) Select units to control by either clicking directly on the map or by using the hierarchy interface;
(2) Begin order mode via a button on the interface;
(3) Click on the map on the location for the units to perform a given action;
(4) On the menu that opens when a location is clicked, select the desired action;

As well as fulfilling the stacking behavior 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 inside the circle (b)) informing of the situation.

In order to illustrate the major details of the proposed architecture, two charts were made. The first presents the interactions between the whole system and the clients during a session and the second shows the different modules inside of the system and the relations between the client and themselves.

The proposed architecture will be able to create a realistic military simulator which can be used for the training of army officers.

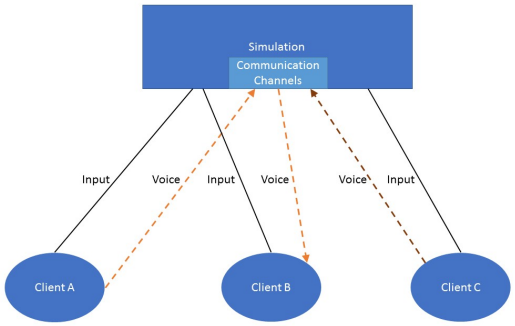

Figure 5: Example of multiple clients connecting to the simulator. Each color of the dashed line represents a different channel. When client A sends a communication, client B, that is on the same channel, receives that communication. Client C is in another channel, so not only does it not receive the first communication, but also no client receives C's communication.

As the proposed architecture is highly complex, in order to test our architecture, 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.

7 SOLUTION IMPLEMENTATION

This section describes the POC that was implemented to demonstrate the feasibility of the model described on the previous section.

Having analyzed 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 surges during development;
- There is a large quantity of *Unreal Engine* tutorials online;
- The *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.

This POC's objective is to implement a simplified version of the proposed architecture in order to demonstrate its potential to create a lower priced military simulator.

The developed POC is not as complex as the proposed military simulator, but it suffices to demonstrate the possibilities that a game engine such as *Unreal Engine* offers for this kind of project. Given the short timetable for the development, *OpenEaagles* was not used, as the time that would be invested in understanding that complex framework could not be spared.

And so, to demonstrate the capabilities of *Unreal Engine* , the developed POC focused in implementing features in three of the modules discussed in section 6.1, namely the Simulation Interface, the Simulation Server and the Unit Database.

7.1 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.

7.2 Doctrine

As stated in Section 6.1, the doctrine will be used to customize the behavior of the AI controlled units, and so, separate the unit's implementation from its behavior 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 the concept of a doctrine. Each class in this list contains one or more references to the one next to it.

(1) DoctrineInfo - Contains the distinguishable information for a doctrine, like its name. Entry point to the decomposing process for the orders, passing custom orders to the next level and invoking directly either basic or general orders on the MasterUnit instance who is executing the order.
(2) UnitOrders - For each unit that is assigned custom orders via the editor, an instance of this class is saved. Organizes the various orders for this unit by name.
(3) OrderInfo - Saves information of a specific order, like its name and different action sets. At this level, the action set which will be executed is chosen, with the criteria being both its condition and restrictiveness (between two passing conditions, the action set whose condition is most restrictive is chosen);
(4) ActionSet - Contains references to the actions which belong to each of the subunits inside the action set, along with the condition associated with the Action Set.
(5) Actions - Wrapper class for an array for the actions which constitute all the a subunit's action for the actionset
(6) Action - Class which translates the received or saved parameters into parameters for the next level of the decomposition.

Each time the action class decomposes an action, a new decomposition process will start at the DoctrineInfo level, until that class is called with a basic action which, as it cannot be decomposed, will signal the end of the invocation of an order. During the decomposition process at the action class, if there is an interrupt condition attached to the action, it is added to unit's blackboard, and checked via a service until either the condition verifies itself and the order is skipped to the next one, or the order continues to its supposed end.

These classes are generic to the degree that, regardless of the actions that are added to the doctrine in its editor, the decomposition process will be executed as expected.

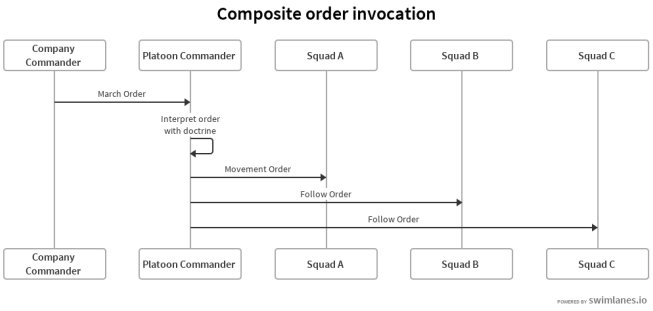

Figure 6: Invocation of a composite order

In figure 6 it is possible to see an example of an invocation of a composite order. The commanders can either be humans 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.

A doctrine is attached to a faction during a simulation and can be easily changed in the scenario editor. This allows to easily test different doctrines for the same situations, understanding to what degree a certain military doctrine is impeding military objectives to be achieved to the fullest.

In the next Section, we will discuss what was created in order to show that the model proposed in this thesis can define a realistic military simulator with a lower cost than those available on the market.

8 VALIDATION

In this Section, we will describe how we tested our POC, how we prepared the tests, how were those tests conducted and their respective results. As the POC was created to show the potential that our proposal has, we created two different sets of tests. The first set was to test the current interface of the POC and the second set was to see if the current internal calculations of the simulation could be deemed as realistic, in the context of this thesis.

8.1 Metrics Used

As stated before, the tests that were made were divided into two different sets, with different objectives namely one to test the interface and another to test the simulation's realism. The first of these sets was further divided into three phases, each one to evaluate a different module of the POC.

At the end of each phase of each set, the users would have to complete a questionnaire related to their experience with the simulator during that phase.

8.1.1 System Usability Scale. The System Usability Scale is a standardized form in order to measure the usability of a system. It is made from a question with 10 sentences, which are then classified by the user with a 5 degree scale, with the lowest value attributed to Strongly Disagree and the highest value to Strongly Agree [6].

The benefits of using the SUS are:

- The SUS is easy to perform by the users;
- Even with a small sample of results, its results are reliable;
- It can differentiate between usable and unusable systems;

More specifically, the SUS is constructed from these sentences:

- I think that I would like to use this system frequently.
- I found the system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in this system were well integrated.
- I thought there was too much inconsistency in this system.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I needed to learn a lot of things before I could get going with this system.

After the user evaluates each sentence from 1 to 5, using the before mentioned scale, the input is used in specific calculations to convert that initial score into a scale of 0-100 [5].

More specifically, the process is:

(1) For each of the odd numbered sentences, subtract 1 from the score;
(2) For each of the event numbered sentences, subtract 5 from their value;
(3) Add all these numbers from the previous calculations;
(4) Multiply the result by 2,5.

By converting the initial results into a scale of 0-100, the results are simpler to interpret [5].

The results are usually interpreted in the following manner:

- If the end result is higher than 80.3, then the system has a high usability;
- A score between 80 and 60 indicate a system which has some usability, but it could be improved;
- A score under that value expresses a system which needs to improve the usability substantially.

8.2 Experiment Preparation

As the constructed POC has three modules with a flow between them, as required information is created through the Information Editor, the sessions are prepared through the Scenario Editor with the previously defined information and then that scenario is used by the Simulation Server to create a training session, we created the tests so that the user would have to follow that flow.

More specifically, the first set of tests had the objective of understanding the current state of the interface and so the tests would have to encompass all the modules which were implemented in the POC. As it relates to interface, this set could be done by anyone.

The second set of tests had the purpose of evaluating the realism that the current POC offers in terms of military behavior and as

such could only be performed by persons which had some degree of understanding of military tactics.

8.2.1 First set of tests. More specifically for the first set of tests, the users would follow the flow of the program, beginning by using the Information Editor to create a TOE with two basic units (an infantry squad and an infantry headquarters) and two composite units (an infantry platoon and a company infantry). The two basic units would be related to the infantry platoon and the company infantry would be composed by infantry platoons.

Using that information, the user would then use the Doctrine Editor to create two orders, one for each composed unit with some degree of complexity, with different ActionSets and by having actions depend on each other instead of being simple flows of actions for each subunit.

After finishing with the Doctrine, the user would have to create two different formations, creating them from the defined composed units.

When the user completed this first section of the tests, he would answer to a questionnaire, evaluating his experience with the interfaces of the Information Editor.

The second section of tests was directed to evaluate the interfaces of the scenario editor. To show the integration of the different modules of the simulator, the user would use the TOE, Doctrines and Formation built in the previous section. The user would change the information associated with the Factions, changing their name, TOE and Doctrine that it would use. After associating the TOE and Doctrines, the user would place the Formations, one in each faction, and place them on the map. After finishing the placements, the user would associate an initial order with one of the squads of one of the subunits, so that in the next phase the user would be able to evaluate how the initial orders are presented to the user. Similarly to the previous phase, the user would fulfill a questionnaire asking about his overall experience with the Scenario Editor's interfaces.

The third phase was to use the created scenario in the previous sections, to show the user the various elements available to the user during a simulation, like the hierarchy, objectives and messages interface and the process to give orders to the units. By choosing to play as the commander of the unit, which has a subunit with initial orders, the user would see the how those initial orders are translated into a simulation. After fulfilling those initial orders, there would be tests related to the usage of the interface, namely selecting a unit via the hierarchy interface and giving orders to its controlled units. After finishing the objectives of this phase, the user would answer to a questionnaire, to offer feedback on the simulation interfaces.

Results- Our testers were students at IST who were used to working with a computer and usually played videogames, which would make them good candidates to test a computer program. As this thesis's objective is to create a military simulator, its users are supposed to understand and apply the military concepts which were discussed previously. As such, the results of the tests were affected by this lack of knowledge.

The SUS results of the various implemented modules were:

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

Analyzing the results given by the various questionnaires and the SUS, it can concluded that the interface of the system is one of its weakest points, as it was never the focus of the development. That resulted in a POC which, despite implementing various complex systems and functionalities, is not easy to use and thus, the interface in the real version of the application should a item of major importance during the development.

8.2.2 Second set of tests. The second set of tests is dedicated to evaluate the realism associated with the created simulation. As the POC is of a military nature, this set of tests was only done by people with knowledge of military tactics and behavior. 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, similar to the training done at the Military Academy . The used scenario was prepared with the help of Major Ricardo Silva of the Portuguese Military Academy and it placed a Light Infantry Company belonging to "Faction A" versus a Light Infantry Platoon belonging to "Faction B" at the Military Academy installations, where the trainees controlled the units from "Faction A" and had to destroy all the units belonging to the "Faction B". Besides these commands, the trainees' professor was playing the role of umpire, assigning orders to every unit present on the map, commanding directly all the AI-controlled units and sending orders to the human-controlled ones.

After finishing the tests, as they were numerous, we would collect their statements regarding the simulator in general, emphasizing the best and worst aspects of the model.

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

In particular, three officers came from the Military Academy: TCOR. Jorge Ribeiro, Major Énio Chambel and Capitão Hélder Clemente. We initially did a presentation regarding the thesis as a whole, focusing on its architecture and how it was applied to the POC.

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

- The program is simple to understand and manipulate, reinforcing what was pointed out before about how the understanding of military concepts is important to understand the program and how it functions;
- The program is able to 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 is able to output all the information about the terrain which is relevant for them, like, for example, 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 behavior follows a doctrine, which is very important in a simulator.

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

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

9 CONCLUSIONS

In this document, 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, it is adequate that only a proof of concept of this system was 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 the engine 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 Chapter 7, it is possible for the units to belong to any echelon and thus different scales of command can be simulated. The adaptable behavior that the units should have is supported with a doctrine as described in Chapter 7. More specifically, this implementation allows for the creation of orders for any unit (from the platoon level to the upper echelons) by defining orders for a given unit using the orders defined to 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.
Using a game engine, like *Unreal Engine*, offers the server-client architecture that is required. Through *inheritance* in classes, we can easily create different roles for the clients, by implementing the common functionalities in a master class and then creating a subclass for each desired role. In regards to 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. We also implemented a simplified version of a scenario editor. In regards to the requirements not contemplated in this paragraph, they are reserved for future work.

The results obtained from the tests demonstrate that our hypothesis (that 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. However, the interface tests demonstrate that it should have received a greater emphasis during development. 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.

10 WHAT WAS LEARNED

During the development of the POC various difficulties were encountered. These had to do with the overall unexpected complexity of the project, a lack of knowledge of the *Unreal Engine*'s architecture and the fact that the more advanced features in *Unreal Engine* , namely in AI, are not available when using *blueprints*. One of the biggest disadvantages of using this system was the fact that on map structure, which associates a key with a value, the values could not be an array, which difficulted the development process. This, as well as other problems, would lead to an overall lack of efficiency of the code and an interface with some issues, as those identified during the tests. While the developed project shows that the proposed model can fulfill the desired objectives, some of its implementation would have to be rethought as it was influenced by the limitations of *blueprints*. For example, the AI Navigation algorithm used by *Unreal Engine* , the A*, is not customizable in *blueprints*, which made the heatmap calculations costly, as we had to make traces on the map to determine the affected area. The lack of understanding of the *Unreal Engine* 's architecture lead to many problems related with replication, which would culminate in it failing on the day of the tests with the officers from the Military Academy.

11 FUTURE WORK

While it was possible to create a POC which demonstrated 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, and thus, more portable as well as more stable. As stated at the beginning of this Chapter, the work focused mainly on both the architecture and the unit's requirements, as they were more important for the purpose of this thesis, leaving the rest of the requirements for future work.

REFERENCES

[1] [n. d.]. What is a Table of Organization and Equipment? ([n. d.]). https://www.globalsecurity.org/military/library/policy/army/toe/toenum.html
[2] André Cunha. 2011. *O emprego do sistema de simulação construtiva como ferramenta de apoio à decisão: uma proposta ao exército brasileiro.* Master's thesis. Escola de comando e estado-maior do exército, São Paulo.
[3] Aaron P. Jackson. [n. d.]. The Nature of Military Doctrine: A Decade of Study in 1500 Words. *The Bridge* ([n. d.]). https://www.realcleardefense.com/articles/2017/11/15/the_nature_of_military_doctrine_a_decade_of_study_in_1500_words_112638.html
[4] João Santos. 2012. A Simulação. Contributos para a formação e treino.
[5] Nathan Thomas. [n. d.]. How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website. https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/.
[6] usability.gov. [n. d.]. System Usability Scale (SUS). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.
[7] Erin Wayman. 2012. What Is War Good for? Ask a Chimpanzee. (2012). http://www.slate.com/articles/health_and_science/human_evolution/2012/10/chimpanzee_wars_can_primate_aggression_teach_us_about_human_aggression.html