TÉCNICO LISBOA

# Military Confrontations Simulator for the Training of Army Officers

**Tiago João de Oliveira Furtado Pereira**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisor: Prof. Pedro Alexandre Simões dos Santos

## Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Pedro Alexandre Simões dos Santos
Member of the Committee: Prof. António Joaquim dos Santos Romão Serralheiro

**May 2018**

# Acknowledgments

I would like to thank my dissertation supervisor Professor Pedro Santos for his guidance and support throughout the development of this thesis. I would also like to thank all the army officers who helped during the development of this thesis.

I would also like to thank both my parents and my girlfriend for all their support in the most difficult times during the creation of the thesis and of the proof of concept.

I also acknowledge the support of the *Unreal Engine* community, particularly Victor Burgos, for helping me deepen my understanding of system's inner workings, making the creation of this thesis possible.

Last but not least, to all my friends that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

# 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 presented. 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

# Resumo

Neste documento é apresentado um estudo sobre os requisitos para desenvolver um simulador militar para o treino de oficiais do Exército. Por forma a compreender quais esses requisitos, realizaram-se entrevistas com oficiais do Exército responsáveis pelo treino de outros oficiais e foram estudados jogos de guerra das diferentes variantes (tabuleiro, jogo de vídeo e simulatores militares em uso por forças militares no presente). Deste estudo resulta uma comparação entre os vários simuladores no mercado e um conjunto de requisitos para um simulador militar. Também foram estudadas *frameworks* que permiteriam o desenvolvimento de simuladores militares e que têm um baixo custo ou que são grátis. A partir dessas duas análises, é apresentado um simulador militar desenvolvido através de ferramentas de desenvolvimento baratas, sendo que a vantagem deste simulador é ser mais barato que os disponíveis no mercado. Por forma a demonstrar as capacidades das ferramentas de desenvolvimento neste contexto, foi criado e testado um protótipo. A partir dos resultados dos mesmos, pode ser concluído que a arquitectura projectada consegue cumprir os objectivos estabelecidos.

# Palavras Chave

Simulador militar; Simulação de baixo custo; jogo de guerra; Confrontos militares

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acronyms

**AI** Artificial Intelligence.

**BT** Behavior Tree.

**DIS** Distributed Interactive Simulation.

**DSS** Decision Support System.

**GIS** Geographic Information Systems.

**GML** Geographic Markup Language.

**HLA** High Level Architecture.

**HTN** Hierarchical Task Network.

**POC** Proof of Concept.

**RTI** Run-Time Infrastructure.

**SOP** Standard Operating Procedures.

**TOE** Table of Organization and Equipment.

# 1

# Introduction

## Contents

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 [31].

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.

## 1.1 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?

[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

## 1.2 Objectives

The present work 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 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;

## 1.3 Thesis outline

The rest of the document is organized as follows: in the next Chapter we study various military concepts associated with this thesis, the military process to make decisions and our analysis of both wargames and simulators. In the final part of the same Chapter, the requirements for a military constructive simulator are presented, as well as the different frameworks and Artificial Intelligence (AI) concepts which could fulfill such requirements.

Our proposed solution and its architecture is presented in Chapter 3, explaining how the different concepts described in the previous chapters are modeled. The developed Proof of Concept to demonstrate the capabilities of the chosen development tools is described in Chapter 4, justifying the choice of the tool in that Chapter's introduction. In Chapter 5 we present the results obtained from the tests made with the POC and their respective analysis.

Finally, in Chapter 6, we present our conclusions about the developed work and also the further work to create a commercial product based on this thesis.

# 2

# Theoretical Background

## Contents

Our purpose is to present a proposal for the creation of a virtual simulation program for armed forces through cheap development tools, facilitating access to a simulator for training purposes. As it will be a computer program where the simulated entities in the environment are autonomous to a certain degree, therefore controlled by an AI, it is necessary to understand both how the military is organized and their decision making, as well the AI concepts which would enable to modulate such behavior. Furthermore, the concepts of simulation and wargaming are also studied, being described the different kinds of simulation and the variants of wargaming, describing its inception and its contributions for the creation of an accurate simulation. Next, we study what is necessary to create a computer program with the required characteristics, investigating the frameworks which can fulfill this document's objectives, understanding how terrain information is stored digitally and what are the AI concepts essential for the development of a accurate behavior. Finally, we present the requirements for a military simulator.

## 2.1 Military Concepts

This first section describes concepts which are essential in a modern military setting and thus important to be integrated in a computer program to use in that setting. The studied concepts are those of Table of Organization and Equipment (TOE), military doctrines and military decision-making.

### 2.1.1 TOE

Tables of Organization and Equipment are used by the military to describe abstract military units in terms of their mission, their capabilities, their internal structure (connections to other described units) and the required equipment in order to achieve that mission [32].

A TOE provides the military with a uniform structure so that units belonging to same branch of the military (like infantry) all have at its base the same core guidelines [32].

Depending on the army, the concept of a TOE can have different implementations. For example, in the U.S. Army there are four types of TOE:

- Base Table of Organization and Equipment (BTOE) - Describes a unit's structure and needs in terms of both personnel and equipment;
- Objective Table of Organization and Equipment (OTOE) - Update over the unit's BTOE according to new policies or initiatives;
- Modification Table of Organization and Equipment (MTOE) - Modification of a BTOE for a specific unit (but with the same structure) whose needs (and thus its requirements) are different from those described in the BTOE.

- Table of Distribution and Allowances - Temporary TOE assigned for a unit during a mission. It is only used when there is no TOE which encompasses the unit's requirements for that mission.

### 2.1.2 Military Doctrines

While the TOE describes and provides military units with a uniform structure, the military doctrines equips them with a framework for their actions, which comes from the core beliefs of the military organization. However, it does not seek to provide rules for the military units to follow during their mission. Instead it guides their actions, standardizing operations and providing a common lexicon for use by the military planners and leaders [15].

Usually there is some confusion between the concepts of military strategy and military doctrine. Military strategy is defined by NATO as

> presenting the manner in which military power should be developed and applied to achieve national objectives or those of a group of nations [2]

Doctrine is defined by Aaron P. Jackson, an author of various military doctrines, as being

> a tangible representation of a military's institutional belief system regarding how that military understands, prepares for and (in theory at least) conducts military activities [15]

and so, doctrine seeks to answer the following questions regarding a military force:

- Who the military perceives itself to be;
- What is their mission;
- The manner to fulfill that mission;
- How has the mission been achieved since the army's inception;

As figure 2.1 shows, military doctrines come from a specific context, derived from the political and technological realities as well as the requirements of the time, and thus it should be constantly subject to update. Failure to do so may result in an inadequate doctrine in relation to the military's needs and objectives at the time.

### 2.1.3 Military Decisions

In order to build a military simulation, it is useful to understand what leads a human being to a decision and particularly how military personnel make decisions.

[1] Source: https:www.realcleardefense.com/articles20171115the_nature_of_military_doctrine_a_decade_of_study_in_1500_words_112638.html

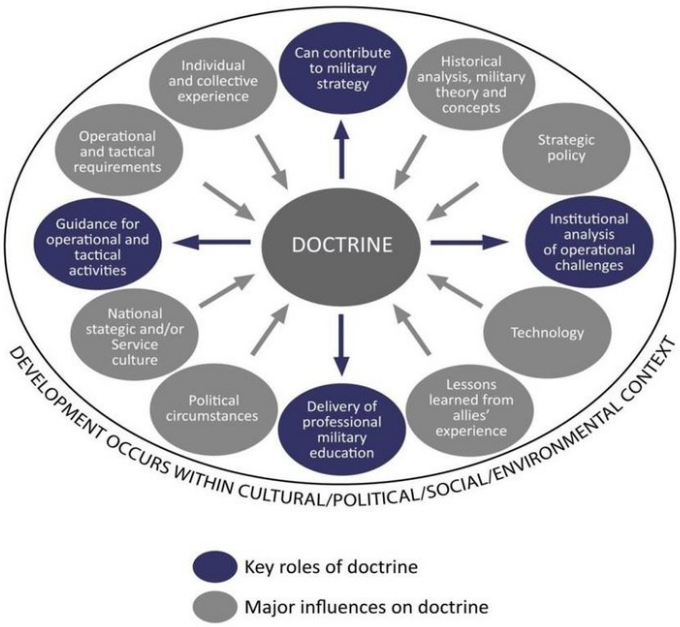



**Figure 2.1:** Chart showing what influences a doctrine and what it influences. [1]

Good decisions are made by exercising good judgment, which involves understanding the situation clearly and then deciding in the most efficient way. Moreover, a person's judgment is directly influenced by their characteristics and experiences. As such, the military commander must have the characteristics that lead him to make good military decisions, like being able to adapt to change and to determine what is an acceptable decision [23].

More specifically about military decisions, the modern battlefield is located in the Era of Information where:

- Ground troops are easily moved;
- The soldiers, when deployed on the field of battle, are trained in order to correctly assess and solve many different situations independently of their commander;
- Social opinion must be considered when deciding a course of action;
- There are more civilian casualties, as a battle is not limited to a field, as in the past;
- It is more likely that combat happens in urban environments;
- There are no temporal and spatial limits to the battlefield;
- The battlefield is very likely to change due to the fast pace induced by current weapon systems.

And so, military commanders are subject to a great deal of pressure, which can easily lead to making a wrong decision, which could result in more casualties during military confrontations [29].

For someone to clearly understand a situation, he or she must have information, which will help to better define the problem and create alternative solutions. Biologically speaking, the human brain tries to match any new situation with an already experienced one, which leads to overlooking the details where it differs [18]. Usually that mechanism is sufficient, and the situation is successfully resolved, but sometimes it is not. When that happens, the brain learns a new pattern. A fault in this mechanism is that if the agent does not recall the exact details of the experience (leading to a faulty analysis of the situation), he or she will not apply that knowledge correctly, leading more likely to a wrong decision [25].

As well as having information, there are other factors to consider a military decision as being a good one:

- **Timing** - As modern combat is something which can change rapidly, a commander must contemplate the time it takes for a decision to be fulfilled by its troops. Timing issues are most likely caused by poor planning, lack of a global view of the battlefield or compartmentalized thinking and could result in an unpredictable development of the war [1].
- **Relevance** is linked with the impact that a decision will have on the battlefield. For a decision to be relevant for a commander, it must influence all of its troops as a whole. For example, when to execute a contingency plan [1].

It can be concluded, by considering the description of the modern battlefield as well as the explanation about the human brain and the its limitations, that commanders cannot cope with today's battlefield unaided. Furthermore, due to the growing complexity of the weapon systems that are deployed by the military, it is necessary for the soldiers to be subject to a more rigorous training. That rigorous training, as well as the necessity of varying the field of training, makes it very expensive. As such, a compromise was found: the usage of realistic simulators in the training process [25]. These allow for the soldiers to gain experience in the usage of the various weapon systems and learning by trial and error, but without having to spend as many resources that otherwise would have to be used. In terms of commander training, these simulators will allow for the commanders to learn in the same fashion, trial and error, but without the consequences associated with a wrong decision on the battlefield.

Now that the role of the simulator in military training has been established, it will now be described what a simulator is and what are its different variants.

## 2.2 Simulation

According to the dictionary[9], simulation can have different meanings:

1. imitation;

2. feigning;

3. sham;

4. representation of the behaviour or characteristics of one system through the use of another system.

The definition which is used in this work is the latter. Simulation can be subdivided in three categories:

- Live simulation;
- Virtual simulation;
- Constructive simulation.

**Live simulation** is one where all the entities, both the people and the systems, involved in the simulation are real. It is usually used to train firearms usage (in shooting ranges) but it can also be used to recreate a scenario to train tactics. It usually involves installing some apparatus in real equipment to mimic the usual resources used by those weapon systems [25]. For civilians, the examples are paintball or airsoft, and for military personnel utilizing equipment like Ultimate Training Munitions (UTM) (as it can viewed on Figure 2.2) which replace actual bullets, while providing an accurate simulation of actually discharging a firearm [19].

**Figure 2.2:** UTM ammunition used during military exercises.[2]

The next item on the list, **virtual simulation** involves real people operating a simulated system. Usually it involves a computer whose input devices are training specific controls (a real cockpit whose windows are computer monitors for a plane simulation). In terms of military usage [25], it allows for the trainee to use the real equipment without the risks of live training, and not being necessary to consume the various resources, like fuel, while training. It is associated with the training of motor skills, like learning to pilot planes, helicopters or ground vehicles. An example can be viewed on Figure 2.3.

[2]Source: http://www.jble.af.mil/News/Features/Display/Article/260780/sfs-train-like-they-fight
[3]Source: https://commons.wikimedia.org/wiki/File:Vehicle_simulator.jpg

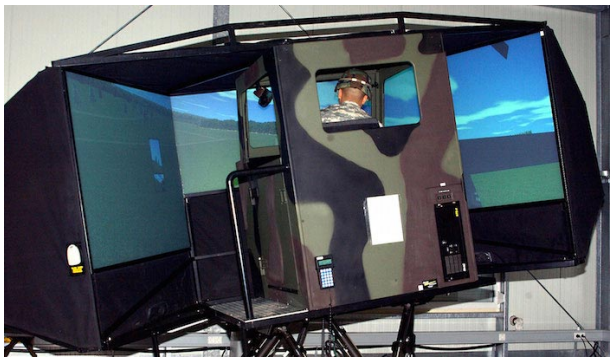


**Figure 2.3:** A vehicle training simulator. [3]

Finally, **constructive simulation**, consists of simulated people operating simulated systems. Users interact with the simulation via inputs to which it reacts, allowing to view the effects of the different actions on the environment. When used by the military [25], constructive simulation is used as a Decision Support System (DSS). These systems are fed with problems and return results in the form of a probability. For example, one can pose a situation where a battalion will fight another one, with a given equipment, doctrine and placements. Will battalion X win? The system then answers with a probability. This is possible due to the computing power available today, that can now simulate the dynamic modern warfare.

To be able to understand what exactly the necessities of a military constructive simulator are, it is necessary to analyze its history and the different aspects of war that are included in these simulations.

## 2.3 Wargames

Representations of war were constructed throughout history in order to better understand it. Chaturanga (whose pieces can be visible on figure 2.4), is one of the first games which represents war in some manner, having pieces to represent foot soldiers, chariots, elephants and cavalry [4].

In today's standards, games like chess (or chaturanga) are not considered wargames as their abstract nature does not serve the purpose of one. The first wargame which was used for military strategy would not be created until the 19th century when a member of the Prussian court, called Georg Von Reisswitz, created Kriegsspiel (an example of a board of Kriegsspiel can be seen on figure 2.5) during the Napoleon Wars.

In Kriegsspiel some elements were introduced which are prevalent even today in wargames:

- Board divided into a grid;
- Real terrain representation;

[4]Source : http://2.bp.blogspot.com/-nuMb9iy4T7A/T_T3qM5zKqI/AAAAAAAAF34/3safkxgYWBw/s1600/chaturanga.jpg
[5]Source : http://general-staff.com/wp-content/uploads/2016/02/Kriegsspiel1.jpg


**Figure 2.4:** The board game chaturanga. [4]

**Figure 2.5:** The wargame Kriegsspiel. [5]

- Randomness for calculating effects;
- Fog of war simulation;
- Communication problems.

The game is played by three players, two for controlling the factions (one for each) and another to be the game-master to calculate the various effects. Other important characteristics which distinguished it from other wargames were the precise scale of troops in respect to the environment (1:2373 in the first version), the usage of tables to measure the offensive capabilities (given the distance) and the fact that every engagement is ruled by a dice, to represent the unpredictability of war. Although Kriegsspiel had a series of new concepts, it was not until the rules were simplified that the game gained some relevance within the Prussian army and one of the main factors to the high quality of its commanders[3]. With Prussia winning a series of wars against opponents which were technologically equal but numerically superior, it became evident that wargaming was a core component of that military success, which led to its global spread. As simulation was now being used by both sides of all major conflicts, the results

obtained by the simulations were taken into account by both sides and as such, those results would influence the troop placement by the different powers, which in turn led to unpredictable situations [3]. As simulation was beggining to be more unreliable, its reputation was negatively impacted.

For example, in a pre-WW1 simulation, the German Empire simulated the invasion of France, by advancing through Belgium and Holland. The results indicated that England would not be able to deploy its forces before the invasion of France was complete, which represented positive chances for the Empire's ambitions. However, on the British side, who had run the same simulation, the results caused the implementation of policies to speed up the deployment of British troops on the continent, if the need should ever arise. As such, when the invasion began, English troops were quickly deployed to France, which would cause the invasion plans to fail, in the sense that it could not be completed before the British arrived [3].

Another example of an event that degraded wargaming's reputation happened in WW2, during the Nazi Germany's invasion of the Soviet Union. The first had simulated the invasion, where the results stated that its troops were capable of severely reducing the Soviet's army size, changing it from 240 divisions to 60 divisions, which would then be forced to spread across a front line from Leningrad to Moscow and deep into Ukraine. With such placements, surely Nazi Germany's army could easily crush the remainder of the Soviet's forces in a swift campaign. However, that did not happen. The invading army did defeat the initial Soviet army, with even better results than expected, but they had not considered the Soviet Union's plan and capacity to train and arm new troops in case of an invasion, which led to the war being prolonged. This extension of the war impaired the Nazi's troops, as they could not sustain both the Russian Winter and the seemingly endless Soviet troops, which led to the failure of the invasion [3].

As it can be seen, wargames lost their reputation because there was:

1. a heavy reliance on the results given by the simulation;
2. a lack of information on the capabilities of the opposing faction.
3. High level of complexity due to the new weapon systems and the social effects linked to every military action;
4. The complexity of the simulators could not grow past a certain point, as the computing power of a computer was not fully realized as nowadays;

These factors led to this first generation of wargames slowly losing its importance for the planning of military operations.

As time went on and new experiments with simulation were made, eventually the USA created the Navy Electronic Warfare Simulator (NEWS), which was a successor of the old tabletop simulators used previously. With the growing development of technology and the USA needing to surpass the Vietnam

War aftermath, the JANUS simulator was developed, creating the first simulator with a graphical interface [26]. With the help of computers, simulators have grown in all of its variants.

Today, simulation is one of the areas that is recognized as having a great importance for resource allocation within the army, as it was made evident that post-digital simulators' level of accuracy has reached a point where they are useful for training or planning purposes.

Having briefly explained the history of wargames, it will next be described wargames in terms of its concept and the different variants that are available.

A wargame is defined as being a strategy game where military operations are being executed. The setting of the game can vary and depending on its type it can be categorized in four different ways:

- Historical;
- Hypothetical;
- Fantasy;
- Science Fiction.

The Historical and Hypothetical wargames are both based on reality in the sense that the units in those games are linked with reality. While Historical wargames deal with conflicts that happened in reality, Hypothetical wargames describe conflicts that did not occur, but which could or will happen. They are based in historical facts which are then modified to create the desired situation. Examples of both these categories are both analysed in Section 2.3.1. Fantasy and Science Fiction wargames are based in either works of fiction or in a setting created for the purpose of the game.

For the purposes of this thesis, only either Historical or Hypothetical wargames are considered relevant and thus, when the term "wargame" is used, what is meant is wargames of either category. Different civilian (or ludic) and military (or serious) wargames existent in the market were studied, in the three different variants in which they exist:

1. Tabletop;
2. Computer Games;
3. Military Simulators.

The main difference between a ludic and a serious wargame is the objective of the game, which will influence the game's pace and its abstractions. In ludic wargames, the objective is for the player to have fun while playing them and in a serious wargame, the objective is to provide an accurate war simulation, reproducing the different realities of warfare.

### 2.3.1 Tabletop

Tabletop wargames are the oldest of the variants. As such, there are many different wargames in the market, with three classifications[33][10], varying on unit, time and spatial scale and the type of objectives. In a descending order (in terms of complexity):

- Strategic - As well as having to command the troops, the games in this category also have some kind of economy system, which determines the reinforcements each player receives in each turn. Normally multiple theaters of war are represented. Unit scale from corps to army size, but also smaller divisions.
- Operational - concerned with moving units within a single theater of war, normally representing a smaller war or a part of a great war. Unit scale is normally smaller than the army size.
- Tactics - Individual units moving on a skirmish, battle or series of battles. The map represented is relatively small, with only some kilometers (km).

However, a single game can belong to different categories. From the great variety of games existent, two tabletop games that possess a certain degree of complexity and represented modern conflict situations were chosen [6]:

- *Next-war: Korea* (NWK);
- *Crisis in the Caucasus: the Russo-Georgian war* (CCRGW).

They will be analysed and then the main conclusions and concepts are presented, to summarize what make these games accurate war simulations. In terms of the scale previously mentioned, they are of a strategic level, due to their unit size, map scale and reinforcement mechanics.

*Next War: Korea* is a game published by GMT games and developed by Gene Billingsley and Mitchell Land. According to the rule book [16], the game is inspired in the confrontation between North Korea and South Korea in the 50s, by posing a continuation of that conflict in the 90s.

In this game, two players face off, with each one having direct control over all of the country's military forces. While this is not entirely realistic, as normally there is a chain of command, it is understandable due to the nature of the game.

The map represents an area around the Demilitarized Zone (DMZ) between the North and South Korea, more specifically 241 kilometers to the North and 402 kilometers to the South, along with its surrounding ocean. It is divided into hexes, where each one represents 12 kilometers. The game is played by turns, where each one represents three and a half days. During his or her turn, a player makes all the desired

[6] This website dedicates itself to board games review. In each analysis, the reviewer places a given complexity score, visible on the respective review page. For Next-war:Korea the score is 4.18 out of 5, for Crisis in the Caucasus: the Russo-Georgian war, the score is 4 out of 5.

actions, waiting while the other player does its actions.
The objective of the game is to collect Victory Points, which are earned when territories are conquered from the enemy. Different types of territories in different situations earn a different amount of points.

There are two ways for players to achieve victory:

- Achieve a certain differential of victory points;
- Conquer the opposing faction's capital;

Victory is not automatically attributed but instead is based on a die roll, where the player rolls a die for Automatic Victory, one for each winning condition that is verified.

*Crisis in the Caucasus: the Russo-Georgian war* is a game published by *K& R Games LLC* and was created by Kyle Cisco. It is based on the 2008 conflict between the Republic of Georgia and the Russian Federation. According to the rule book [5], the game can be played by either one or two players, where each one has direct control over all the country's military forces.

The map is divided in hexes and represents the totality of the territory of the Republic of Georgia, as well as the territories of the Republic of Abkhazia and South Ossetia. It also has some hexes on the other side of the border in the mentioned regions. Each turn is divided into 6 [7] phases, where both players alternately move their units, being able to counter the attacking player's advances. This mechanic allows for the players to experience the stress of war and to realize the importance of having an openness to changing their strategy. Each turn represents 24 hours.

Depending on the scenario being played there are different victory conditions. For instance, if playing the 2008 war scenario, victory is automatically claimed by Russia if they control both Georgian cities of Gora and Tbilisi, its capital, at the end of its turn. Likewise, if Georgia controls Tskhinvali (starts in Russian control) it claims victory. If at the end of turn three, none of the conditions are achieved, the game ends in a draw, as this scenario is limited to three turns.

Now that both games have been summarily described, an overview of the most important characteristics that both games possess that contribute to construct an accurate simulation will be made.

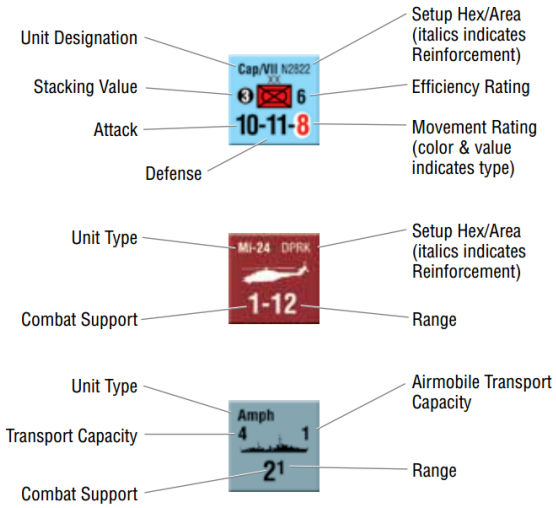
In NWK, the units size varies from Battalion to Army and in CCRGW the units range from Battalions to Brigades. In terms of unit characteristics, NWK is more detailed, distinguishing between the type of unit (ground, air, sea), where each one has several characteristics, which are next presented (Fig. 2.6).

In NWK, land units have the following characteristics:

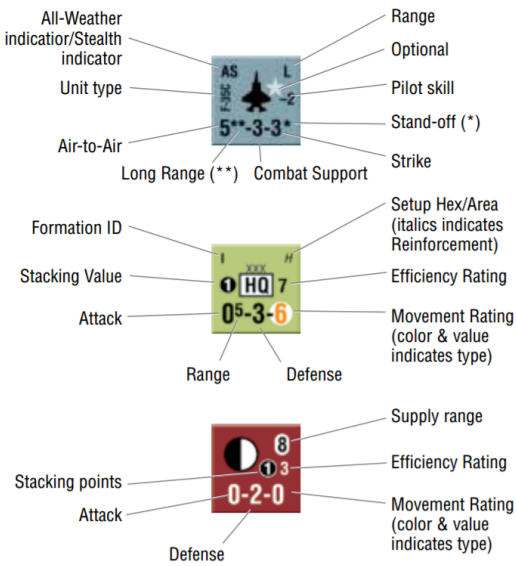
- Attack Strength - Unit strength when it attacks;

[7] 1. Setup Phase (where the Initiative player is determined), 2. Initiative player advanced phase (first attacks), 3. Non-Initiative player limited phase (player moves its units and can engage the enemy), 4. Initiative player main phase (movement and combat), 5. Non-Initiative main phase, 6. End Phase

[8] Source: NWK rule book.




**(a)** Basic counters




**(b)** Advanced counters

**Figure 2.6:** NWK units counters variants according to the two sets of rules [8]

- Defence Strength - Unit strength when it defends;
- Efficiency Rating - represents the unit's morale, training, cohesion and weapons systems. This is used to influence the hit chance in combat;
- Formation ID - number of units represented;
- Movement allowance - number of moves per turn allowed, which represents that unit's mobility;
- Starting Hex;
- Stacking value - space occupied in the hex as hexes have a maximum stacking capacity.

As stated before, air units and sea units have different variables:

- Airmobile transport capacity
  Transport capacity - number of stacking points that the unit can transport;
- Combat support rating - ability of the air unit to support ground units. It translates to a beneficent die roll modifier during engagements;
- Range - effective range (in hexes) in which the unit can support ground troops. It also represents the movement capacity of helicopters;

Other key aspects that pertain to both tabletop wargames and that are important to refer given the objective of the thesis 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.

The problem with tabletop games in today's context is that given the previous description about modern combat and the history described in Section 2.3, although it is possible to learn some aspects that are important to create an accurate simulation, the timing and pace of these games as well as the necessary complexity to simulate modern warfare is no longer acceptable given that there are better methods (presented in the following sections). However, it must be referred that the military uses tabletop wargames, like sand tables (as it can be seen in Figure 2.7), to plan their action and visualize the terrain where a given operation will take place, as it allows for a 3D view in the field [30]. Such tables were also used, for example, by the Iraqi military in order to plan for the defense of Kuwait City[20].

**Figure 2.7:** US army using sand tables to plan a military operation [9]


**Figure 2.8:** AMX-30B2 battle tank description. [11]

### 2.3.2 Video games

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.

In terms of videogames, various examples were reviewed but the one selected, Wargame: Red Dragon, outweighed the features of the others and added others, which made it the most realistic videogame analyzed.

*Wargame: Red Dragon* has 17 factions[10] which serves to show that although the factions follow an archetype, it is important to insert the actual troops you are simulating for both their personal tactics (military doctrine) and equipment (a unit's details can be seen in Figure 2.8). As expected of a game which pertains to be a better approximation of warfare that most other video games, units have natural resistances, weaknesses and strengths.

The battlefield is mutable, which is important in an accurate simulation, shown by the fact that positions on the battlefield can be hit with napalm, artillery or air units, from against which is difficult to mount a defense for, forcing the employed strategy to be dynamic. The game allows for infantry units

[10] All units present in the game are accessible on this website `https://aqarius90.github.io/FA_WG_Utilities/` at the date of writing, 21 of April, 2018

[11] Source: in-game image.

to garrison buildings, which protects them and extends their line-of-sight. In terms of tactics, the game allows to instruct the units to not engage enemy units, to impede them from both attacking and revealing their position, or to command them to adopt a given formation.

As is normal with video games which are classified as wargames, the fog-of-war exists but is not explicitly represented, which means that the game does not cue visually what are the zones effectively seen by the controlled faction. Another aspect that is interesting is that units, as the map is zoomed out, are grouped and are issued the same orders. This allows for a fast switch between micro and macro managing of the units, that even if it is not accurate in terms of military command, it is a feature which is unusual in common RTS games.

In sum, *Wargame: Red Dragon* provides a good war simulation as:

1. It has a great quantity of highly detailed units (1200 units);
2. The units can perform advanced tactics;
3. It shows the mutability of the battlefield;
4. The fog-of-war is not explicitly represented;

Despite its details, this game can not be considered a simulator due to:

- 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 versus a tank in open terrain, which is not accurate;
- Reinforcement is almost instantaneous;
- The weather is not simulated.
- Unit behaviour is not changed by changes in the environment (there is no unit morale);

Many of these problems are linked with the objective of this game: it is intended for a short (30 minutes to 1 hour) playing session, which is meant to be filled with events, and it is meant to be fun, and therefore cannot cross a certain threshold of realism, so the player does not get bored or frustrated.

### 2.3.3 Military simulators

There is one central aspect which separates a military simulator from a video game: a simulator is used for learning or as a Decision Support System (DSS), not for ludic purposes. This aspect will influence the duration of the sessions and the characteristics and behavior of the units.

Wargames which are used as simulators are either virtual or constructive simulators. When used as virtual simulators, they are a training tool to aid in officer's courses. When it is used as a constructive simulator, it is a DSS.

As the analyzed simulators are used by the military today, their reviews are in the State of the Art Section (2.8)

Having established what is a wargame and more specifically a military simulator, the coming section will discuss the requirements for a military simulator.

## 2.4 Military Simulation Requirements

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 categories:

- Architecture;
- Units;
- Mechanics.

The following requirements were obtained from the analysis of a paper[8], where a preliminary analysis of the requirements for a constructive simulator was made, and from interviews conducted both in the Portuguese Military Academy [12], at their simulation center, and at the Institute of Superior Military Studies [13], with the officers responsible for a constructive simulator.

### 2.4.1 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 sim-

[12] Responsible for the training of the lower echelons of military command.
[13] Responsible for the training of the higher echelons of military command.

ulation. 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.

### 2.4.2 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.

### 2.4.3 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.

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 (as seen in Figure 2.9), 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.

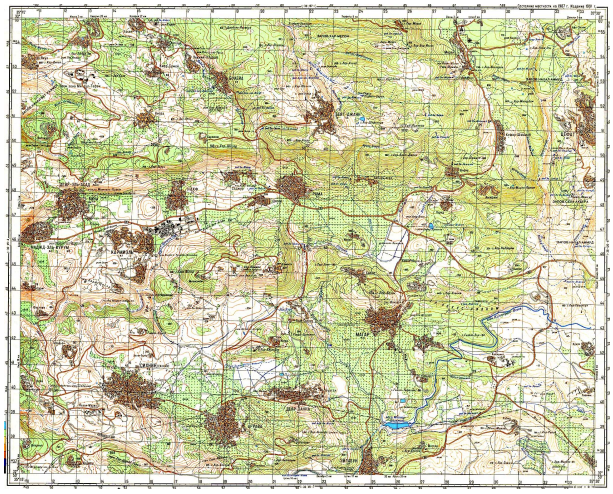


**Figure 2.9:** Example of a military map, with a grid over it. [14]

Now that the requirements for a simulator have been established, the frameworks and AI concepts that can help the process of developing simulators will be analyzed.

## 2.5 Frameworks

In this section we will discuss frameworks which fulfill two requirements:

- Are free or with a low cost;
- Can be used to create a complex system, like a simulator;

We will analyze three different frameworks. Despite not being extensive, this list encompasses a framework which was designed to create simulators and two game engines which have been used to create RTS, of which wargames are a subgenre.

### 2.5.1 *OpenEaagles*

This is an *opensource* framework created to ease and accelerate the development of virtual and constructive simulators. It is developed by a team constituted by Doug Hodson, Chris Buell, Rob Subr, Dave Gehl and Lee Sines. It is written in C++ and C and it has the objective of combining the object-oriented programming advantages with real-time system design techniques. It uses OpenGL [15] and FreeType [16].

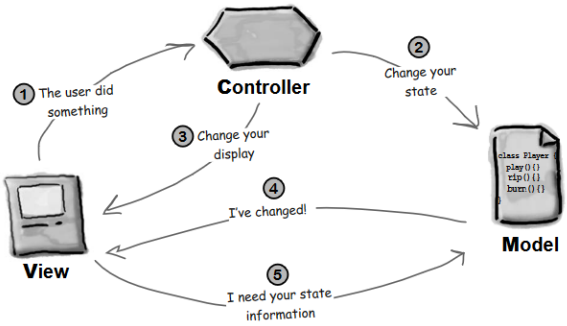



**Figure 2.10:** MVC's organization and inner relations [17]

It is organized into packages, building on the *Model-View-Controller* (MVC) (Figure 2.10 represents the MVC's internal modules relation) software design pattern which divides an application into three interconnected parts so that the internal representation of the application's information (model) does not influence the way the information is presented (view). The controller is the way that the user has to interact with the system and converts any input given by the user into commands for the model or view components. The framework also provides abstract network interfaces, so that custom protocols, like

[15] Library for drawing in C++ or C. Building block of the many graphic systems today. OpenEaagles comes with GLUT which is a library build on OpenGL to aid in developing, by providing many functions premade.
[16] Software to render fonts
[17] Source: Head First Design Patterns, p. 529 (adapted)

Distributed Interactive Simulation (DIS)[18] and HLA[19], can be implemented without interfering with the other modules.

This framework provides the main concepts around a constructive simulator, which is useful as it is not required to build the proposed solution from scratch. However, in order to use the framework, it is necessary to attentively study the documentation, slowing the development process at the beginning, as it is necessary to precisely understanding how the framework works using it to develop the presented solution. Also, as this framework is not widely used, its documentation is limited to what was directly created by its authors, as there is not a community as in the following frameworks.

### 2.5.2 *Unity*

As a game engine, *Unity* is designed to ease the overall process of creating a game, by providing all the necessary tools (images of the various tools can be seen in Figure 2.11), like:

- GUI to organize and access all the project's assets [20] ;
- A Viewport to construct the scene while viewing it;

It also provides with many different features usually found in games like physics (forces, collision detection), Input/Output operations, animations and automatic path finding for the AIs so that the creators do not have to reinvent the wheel when creating a new game. It also provides tools to edit audio assets and a resources usage monitor.

When an object is created different components can be added to it to define it. It is worth noting that besides those mentioned before, we can add scripts written in either JavaScript or C# to the objects themselves, in order to configure the game logic. The same script can then be applied to the different objects, to give them the same behaviour. Following the logic of facilitating the creator's job, *Unity* comes with different classes out-of-the-box for common features in a game, having the creator only to use those classes instead of recreating for each new project. For example, it comes with a network class which is used to handle network connections required by the user, by providing several functions.

In terms of pricing, at the time of writing, *Unity* has different options available, which changes the revenue capacity, the metrics collected from the players or access to other features of *Unity*:

[18] DIS is an IEEE standard for conducting real-time platform-level wargaming across multiple host computers. It distinguishes from High Level Architecture (HLA) because it does not have a central manager (Run-Time Infrastructure (RTI)). Instead, DIS define the Protocol Data Unit, used to establish the connections between the applications [17].

[19] NATO standard which seeks to define a common simulation infrastructure to allow for the interoperability between simulators, regardless of the host's platform. It defines: Rules that the simulation must follow, an Object Model Template which states which are the common objects used by a simulation, with their respective attributes and relationships, and an Interface Specification which allows for the creation of the RTI middleware. The RTI is the central concept of HLA as it allows for the different simulations to communicate during execution [12] [17].

[20] An asset can be anything, like an audio file, an image or a C# or Javascript script

[21] Source: http://www.highpants.net/wp-content/uploads/2013/06/highpants-microsoft-and-unity-game-dev-002.jpg

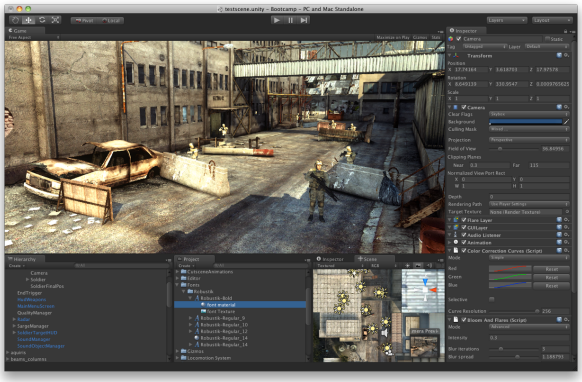


**Figure 2.11:** Unity GUI. [21]

- Personal edition which is free. Allows up to a hundred thousand U.S. dollars of revenue, comes with core analytics and a thousand points of analysis. It allows for twenty players in multiplayer concurrently.

- Plus edition which costs thirty-five U.S. dollars per user/month. Allows up to two hundred thousand U.S. dollars of revenue and allows for two thousand points of analysis. It allows for fifty players in multiplayer concurrently as well as a new skin for the environment, performance reporting of the game and a twenty percent discount on the Asset Store. It also allows to deploy game for every supported platform, like *Android*, *iOs* or Windows.

- Pro edition which enhances the previous plan by removing the limit of revenue and allowing up to two hundred concurrent users in multiplayer. It also enhances the analytics provided, by increasing the points of analysis to five thousand and a forty percent discount to the Asset Store. It comes at a hundred twenty-five per seat/month.

- Finally, the Enterprise edition removes the multiplayer and analytics restrictions. As all the features provided are custom fitted to the client, the price is arranged with *Unity* sales department.

The advantage of using a game engine for creating a simulator is that it is easier to develop as there is a virtual world where the various entities can be directly placed in the world. Another advantage is that *Unity* is widely used and as such, has a large community, which can provide support for the issues that can appear when developing with it, and there are various tutorials available online for creating various concepts within *Unity*.

### 2.5.3 Unreal Engine

Like *Unity*, *Unreal Engine* is a game engine and as such gives the tools necessary for the creation of games. It is developed by *Epic Games* and the main differences from *Unity* are:

- Its internal language is C++;
- There is a graphic way, called *blueprints*, to create the game logic.

More specifically, *blueprints* are the main distinguishable feature provided by *Unreal Engine*.They were created to allow for non-coders to be able to create game logic with visual scripting, as otherwise that logic would have to be implemented through C++. More specifically, blueprints allow for:

- Manipulating materials;
- Creating shaders;
- Creating and editing audio assets;
- Creating the heads-up display (HUD), which is composed with the camera view to form what the user sees in game;
- Creating the different characters in a game, specifying their look and logic;
- Creating the AI contained in a game;
- Creating interfaces [22];

The various editors inside *Unreal Engine* which can be used to create assets are displayed in these images:

In terms of customization of the available nodes, it is possible to create new nodes by coding them in C++ and then annotating that implemention with special macros provided by *Epic*.

In sum, *blueprints* simplify the creation of the various aspects of the game as it is not required to code, accelerating the process to create a prototype, which is one of the objectives of this thesis.

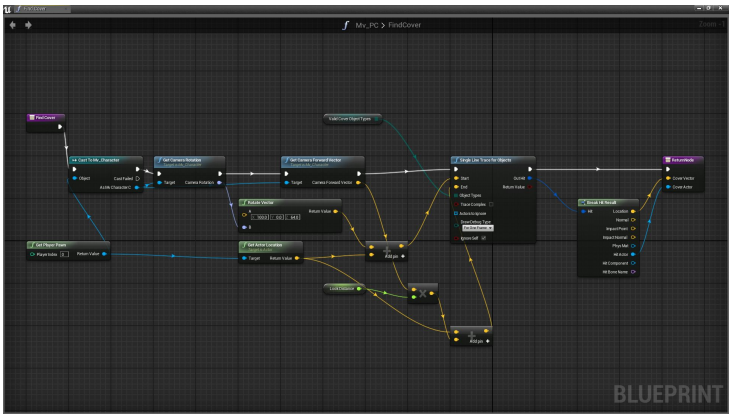
However, it is important to mention that *blueprints* by themselves are insufficient when a certain level of complexity is reached both because the engine's capabilities (particularly in the AI modules) are not available on them and their readability is rapidly decreased as that complexity increases. These limitations are understandable, as *blueprints* are not supposed to replace the C++ coding.

As *Unity*, *Unreal Engine* already brings a plethora of libraries in C++//Blueprints.

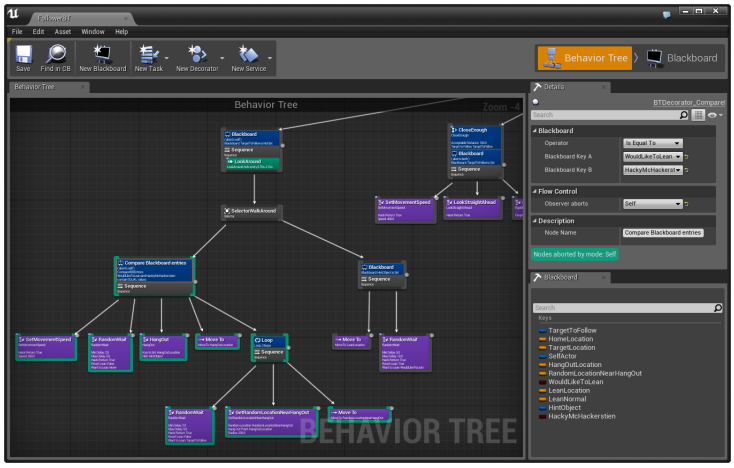
In terms of pricing, *Unreal Engine* is free until the first three thousand U.S. dollars of revenue, per quarter, made by a game. From there, a five percent royalty must be paid to *Epic Games*. For company-specific needs, it is possible to contact the *Epic Game's* sales department to agree on a price.

The coming sections will discuss concepts which are important in order to develop a military simulator, like terrain information and AI concepts.

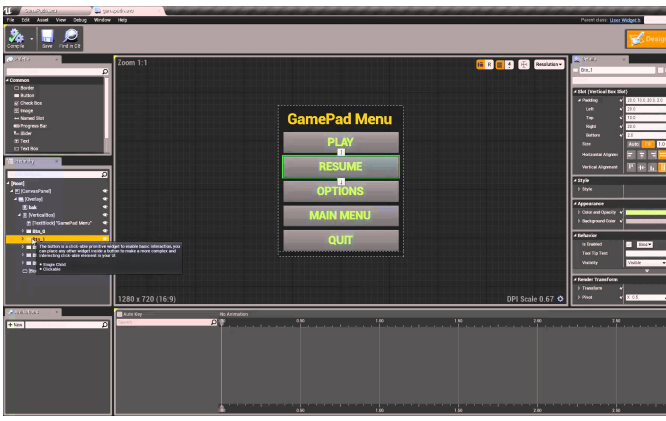
[22] Collection of function definitions to guarantee that a given blueprint can make certain operations.




**(a)** Blueprint to create scripts




**(b)** AI behaviour tree




**(c)** Blueprints to create interface elements

**Figure 2.12:** Various examples of blueprint editors inside *Unreal Engine*. [23]

## 2.6 Terrain data

Cartographic information has grown in complexity and quantity with the development of technologies. One of those innovations was the creation of Geographic Information Systems (GIS). These allow to integrate various technologies in order to create to tool to integrate geographic data and methods for analysis, with the advantage of being able to query large quantities of data in a single database [11].

More specifically, GIS are databases which reference various types of data in a single coordinate system, like latitude and longitude. They offer the means to:

- Add data from different sources such as maps and surveys;
- Store, retrieve and query the data;
- Create reports in various output forms, either in textual (in markup languages, like XML) or visual forms.

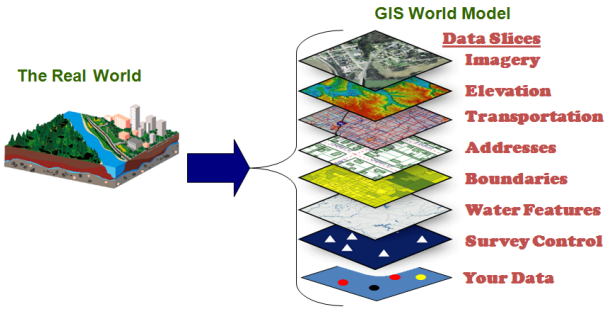



**Figure 2.13:** Geographical information in a GIS for the same region. [24]

For example, for a given region, a GIS can have various maps each one of which them will give specific information about that area, and it organizes them in layers (or coverages or levels). As the information is all referring to the same coordinate system, it will be placed to represent a precisely defined region, as shown in Figure 2.13.

This fact allows for the different layers to be combined into a new layer through the process of map algebra.

For this thesis, it is relevant to extract data from the GIS in a format which can then be fed to the simulator, so that it can use that information in a simulator-specific map. One of the formats in which the data can be exported is Geographic Markup Language (GML)[7]. It is constructed by two components, a schema and a main file. The schema describes the entities used in the main file which has the actual data about the map. GML is usually organized in geometric objects and features. Geometric objects are entities constructed from the different primitives defined in *GML* such as *Points*, *LineString* and *Polygons*, which allows us to profile what exists at a given location. Features describe physical entities existing in the space defined by the geometric objects.

The final concepts which will be explained are Behavior Trees (BTs) and Hierarchical Task Networks (HTNs), both notions of AI which can be useful to model the behavior of real military entities.

## 2.7 Artificial Intelligence

Artificial Intelligence is a discipline which seeks to design and create intelligent agents [22]. An agent is something that acts within in a given environment and context. A agent controlled through AI's concepts can be simply be referred as AI. The degree to which a certain AI's behavior is appropriate depends on its goals, where the way it fulfills them must change according to its perceptions of the environment and its past experiences [22].

For the purpose of this document, we will study two different concepts created by this discipline: Behavior Tree and Hierarchical Task Network. These concepts are used as they allow to model behavior for AI agents.

### 2.7.1 Behavior Trees

A BT is a mathematical model which model a given complex behavior from the usage of other behaviors or tasks where the main BT does not depend on how those are implemented, it just expresses how those are chosen and their sequence of execution [6].

BTs are usually represented as a directed tree and its nodes can be of three types:

- Root node;
- Control flow nodes;
- Execution nodes (tasks);

Each type differs in its position on the tree and its capabilities: the Root node is placed at the top of the tree and has a single child node and no parents, a control flow node has one parent and at least one child node and the execution nodes have one parent and no children [6].

When a BT is being executed, it sends ticks (signal to execute a node) from its root to the children. The children will return one of three statuses after receiving a tick:

- Running;
- Success;
- Failure.

These statuses, in conjunction with control flow nodes, will dictate the manner in which the behavior tree's executes. Control flow nodes can be of two types:

- Selectors;
- Sequences.

They both execute their subtasks in a specific order, the difference being in the criteria used to switch to the next subtask: in a selector, when a subtask returns success, then the selector succeeds and does not execute subtasks after the one that succeeded. In a sequence, it is the reverse: it will continue to run subtasks as long as they do not return failure [6]. A sequence only succeeds if all of this subtasks succeed.

The disadvantages of using behavior trees are the necessary work to translate the complex behaviors into the language through which a behavior tree is defined.

An example of a BT can be seen in figure 2.14, where the $\varnothing$ represents the tree's root, $\rightarrow$ represents a sequence and *?* represents a selector. The circular objects are events and quadrangular objects are

[24] Source: Wikipedia.org

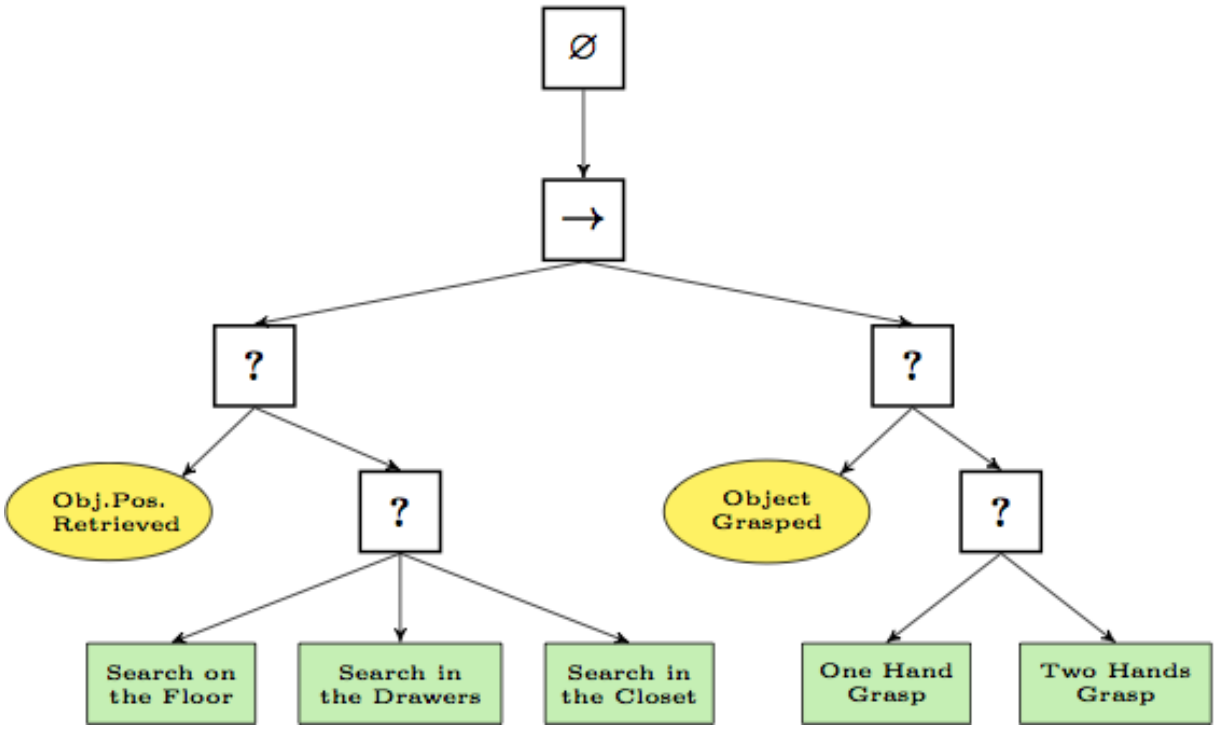



**Figure 2.14:** Behavior Tree modeling the search and grasp plan for a dual armed robot

tasks. As it can be seen, model a behavior like a search and grasp behavior is rather simple. The tree is executed from left to right, so that the robot first searches for the object and if it finds that object it will then try to grasp it.

As such, using these BTs it is possible to model pre-defined behaviour (like the basic and general actions defined in section 3.1). However, to create the behaviour that a hierarchical structure requires using only BTs would make them unnecessarily complex and inefficient given that there is another concept, HTNs, which can help solve the issue in a more efficient manner.

### 2.7.2 Hierarchical Task Networks

HTN are a method through which a set of tasks is executed. While this seems similar to the objective of a BT, in an HTN these tasks are hierarchically dependent. Tasks are divided into three subcategories:

- Primitive tasks - can be executed directly if all its preconditions are supported on the current state;
- Compound tasks - which are sets of simpler tasks (either primitive or compound):
- Goal tasks - task composed of a set of conditions;

Each task can be associated with constraints that specify exactly when a task can be executed in terms of order of execution with the other tasks [24].

A solution to an HTN problem is represented as a set of executable primitive tasks which were obtained from decomposing compound tasks into corresponding sets of simpler tasks [24].

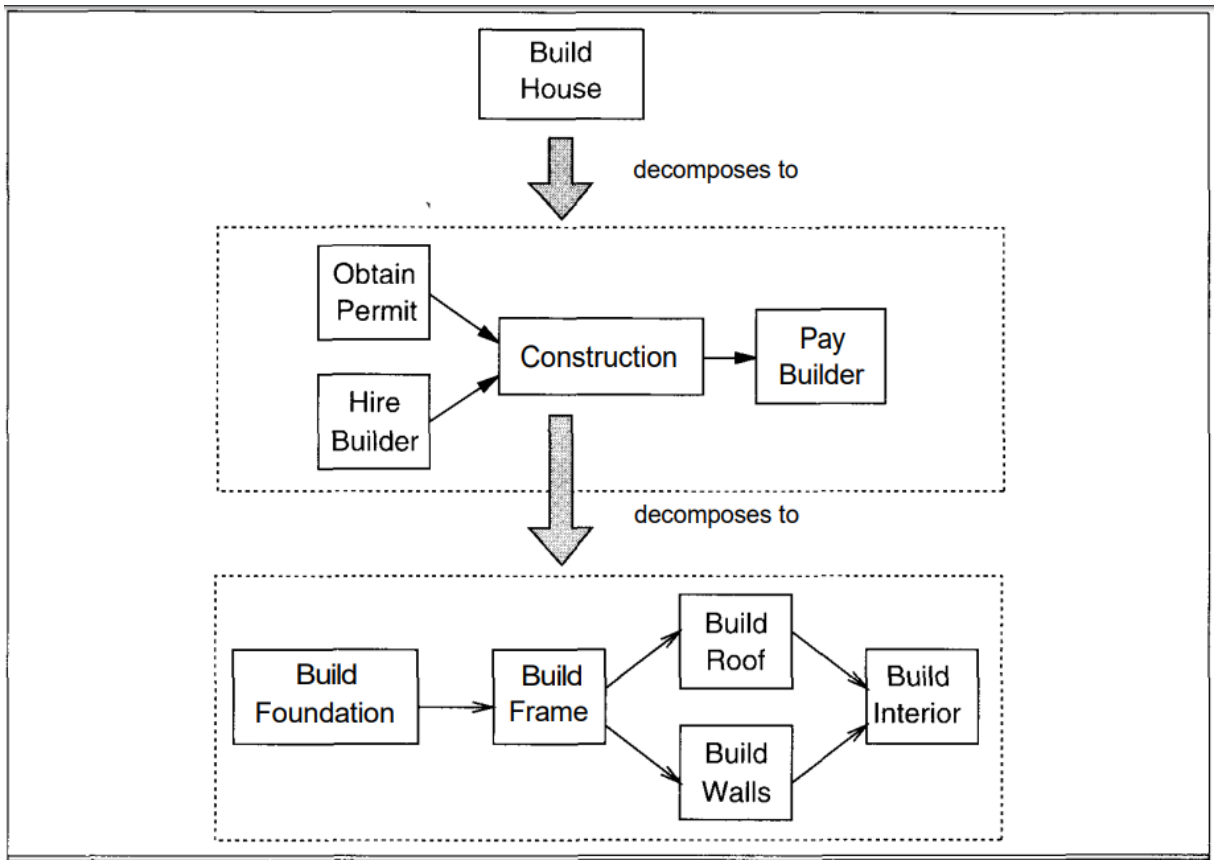



**Figure 2.15:** Example of a HTN problem and its solution

In Figure 2.15, it is possible to see an example of a HTN problem. The example models the problem of building a house by decomposing it in simpler tasks in each frame.

The concept of an HTN is useful for a military simulator as by definition, a military force is organized hierarchically. Orders which are given to a echelon are translated into other orders to the lower echelons, similar to the decomposition process used to reach a solution in an HTN problem. For example, when a company commander issues a march order for a platoon, the manner in which that order is transformed into orders that the units that compose the platoon can execute can be defined through a HTN solution.

The next section will continue to analyze what is essential to create a military simulator, by analyzing the simulators currently used by armed forces around the world.

## 2.8 State of the art

This section contains the reviews of the military simulators. They were chosen for their relevance in military forces as tools for officer training.

In total, four simulators were analyzed:

- *Tac Ops 4*;
- *Sword*;
- *MÄK CST*;
- *VBS Tactics*.

*Tac Ops 4* is the oldest (1994) of the simulators that were analyzed, which reflects on its appearance. However, it has some important characteristics like the characterization of the units and the different orders which are available to give those units, as will be next described. It was developed by a single person, Major I. L. Holdridge and published by *Battlefront.com* [14].

*Tac Ops* is played by turns and each turn is divided into two phases:

1. Orders phase - all players can give orders to their respective units;
2. Combat phase - the orders are executed (1 step).

These phases can be set to be timed, for a better simulation. When using the simulator, the player usually is a battalion or regimental commander.

A fact that distinguishes *TacOps* as a simulator from a video game is the system's objective: *TacOps*'s objective is to provide a realistic experience instead of a ludic one.

As each shot can have a devastating effect on the troops, it is important to place them correctly, so that they are not easily spotted by enemy units or, if spotted, not easily hit. Like in *Wargame: Red Dragon*, the fog-of-war is not explicitly represented, the units that can be viewed by friendly units being the only ones seen by the player. The shots which are fired are not explicitly represented, only lines between the units that made contact.

Artillery is an important component of modern combat as it can rapidly change the battlefield, and *Tac Ops 4* allows for the construction of fire orders, being possible to choose its ammo, between High Explosive (HE) or Armor piercing (AP), location and number of salvos. It is also possible to set if the fire was planned or not, which influences the time it takes to fire, to simulate the necessary calculations done by the artillery crew before firing. In a similar way, it is possible to configure air missions to strike on important positions, having to wait while the air units do not reach their target.

[25] Source: in-game image.


**Figure 2.16:** Example of the SOP screen in Tac Ops 4. [25]

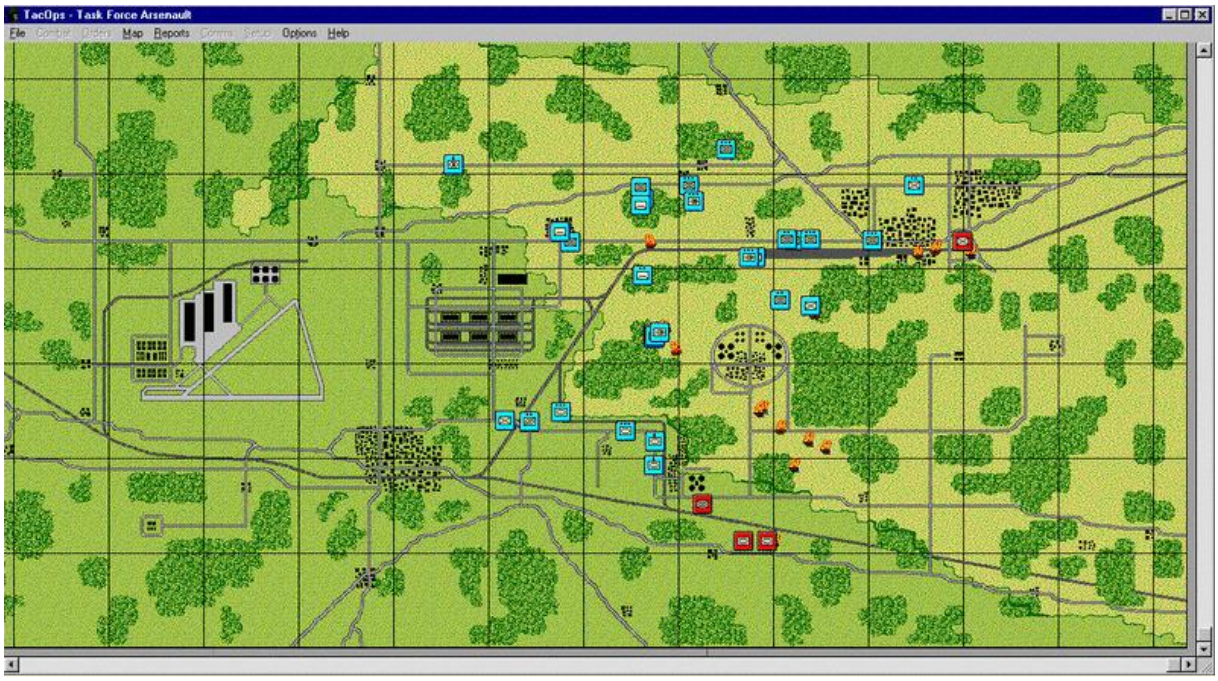
After each scenario is completed, it is possible to review the various actions taken by the players, by watching a recording of the units and their actions.

The simulator supports different numbers of players. It can either be played in solitaire, or with up to 19 players, divided in up to 8 teams. Each client is connected through *TCP/IP*, either from the Internet or in a *Local Area Network* (LAN). The host plays the role of umpire to whom other 19 clients can connect, who are divided between players and spectators. The umpire can introduce new objectives or add new units to any faction mid-game, as well as control any faction's units. It does not support connections to other simulators.

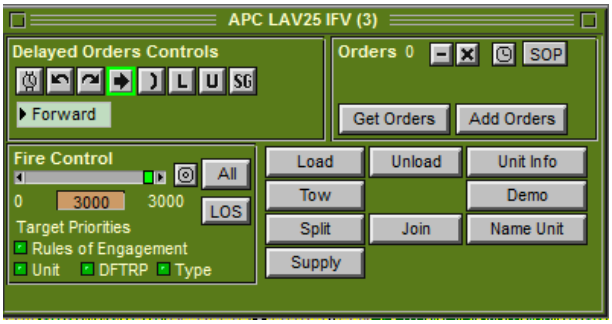
The interface (visible in Figure 2.17), has some limitations, that are understandable given the constraints in terms of computing power at the time, 1994. More specifically, the interface mostly shows the map, and all the different options are only accessible through the toolbar at the top. The orders are given to one individual unit at a time, by double clicking them. That action causes a menu (visible in Figure 2.17 (b)) to appear, where the orders can be altered for that specific unit. Overall, the interface is tiresome and time consuming to use as it just allows for orders to be given through these forms and it does not support keyboard shortcuts.

***VBS Tactics*** is developed by Bohemia Interactive Simulation and it is an interface for the main product of that company, Virtual Battle Space (*VBS*). *VBS* is a simulation server (creates virtual environments populated by AI controlled entities) and it is used in their other products, virtual or constructive (*Tactics*) simulators and in their game series, ARMA. *Tactics* provides the perspective and controls appropriate for a constructive simulation, using *VBS* for the simulation calculations.

[26] Source: in-game images




**(a)** Main screen of TacOps4




**(b)** Order menu

**Figure 2.17:** TacOps 4 interface examples. [26]

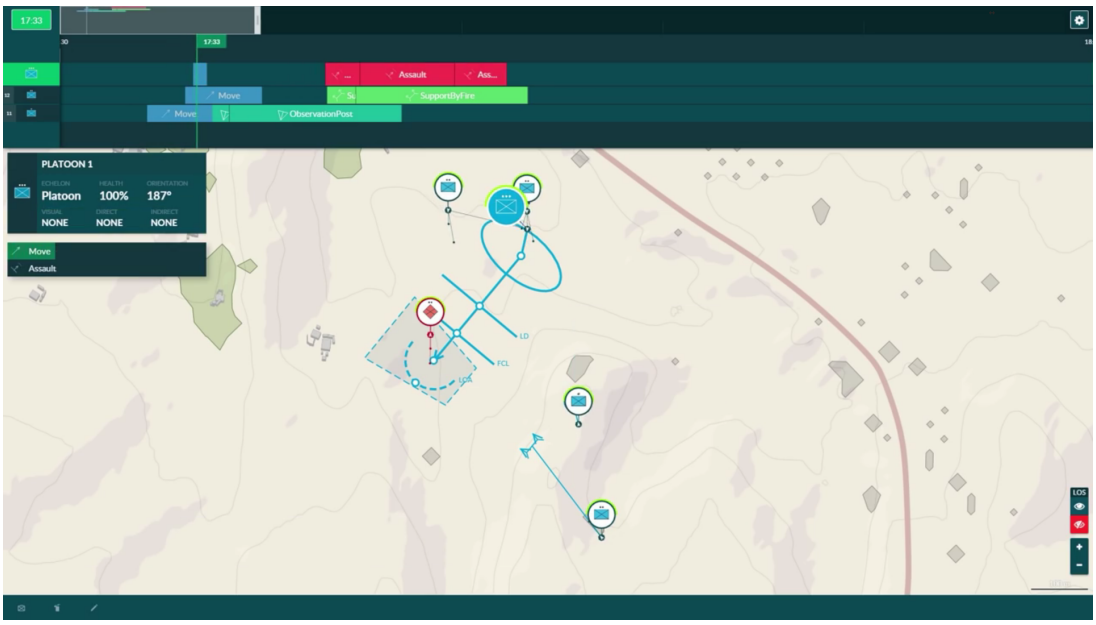
The simulation provided by *VBS* is presented in real time and as expected from a military simulator, the units behave accordingly to military doctrine, establishing positions and engaging the enemies, providing an accurate simulation as real soldiers are assumed to behave as detailed in military doctrine. The orders that can be given through *Tactics* can be scheduled on creation and then can be changed or canceled through the time line (visible in the top of Figure 2.18), making *Tactics*' interface very intuitive and simple to learn (contrarily to the Tac Ops' one). The orders are fulfilled by the troops which are either controlled by an AI or by real people (in virtual simulators). The enemy troops normally are controlled by a human (instructor). After each battle, *Tactics* offers the possibility to review the action in order to debate the choices made during the engagement and further develop the learning possibilities.

In terms of roles, *VBS* allows for players connecting via *Tactics* to join as:

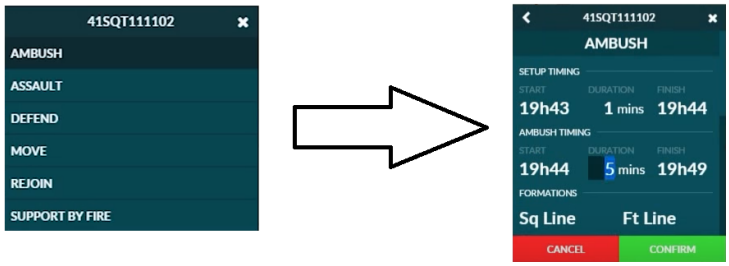
- Trainees, which command a unit of soldiers, ranging from a squad (8 to 12 soldiers) to company leader (80 to 250 soldiers);
- Trainer, which has monitoring features and animation (spawning and control units) capabilities.

It has an internal communication system through which the players can communicate either 1-to-1 or via a simulated radio, with different channels. The communications can be listened to in the After-Action Review.

It implements HLA/DIS standards so it can connect any kind of simulator (real, virtual or constructive) to the created environment.




**(a)** Interface




**(b)** Context menu flux

**Figure 2.18:** *VBS Tactics* screenshots. [27]

The simulator is played from a top view in 3D, where the map has relief as well as height lines (Fig. 2.18). To command the units, the player selects the unit and clicks on the location where the unit must

[27] Source: https://www.youtube.com/watch?v=Ap3jUnePRBU (adapted).

perform an action. The possible actions are displayed in a context menu at that moment, where it is possible to customize the hour at which the action starts and see when will the units be ready.

In terms of architecture, *VBS3* (the most recent version at the time of writing) can function either by having a dedicated server or by one of the clients functioning as client and server. In the first case, the clients join the server, where one of them will be an administrator of the server (logs in as administrator) and will start the scenario. In the other case, the clients join the server and the administrator is the host. Either configuration can be used between distant clients/server or in *LAN*.

To prepare military commanders to official symbology, *VBS Tactics* uses standard 2525C symbology, which can be seen in Figure 2.19.

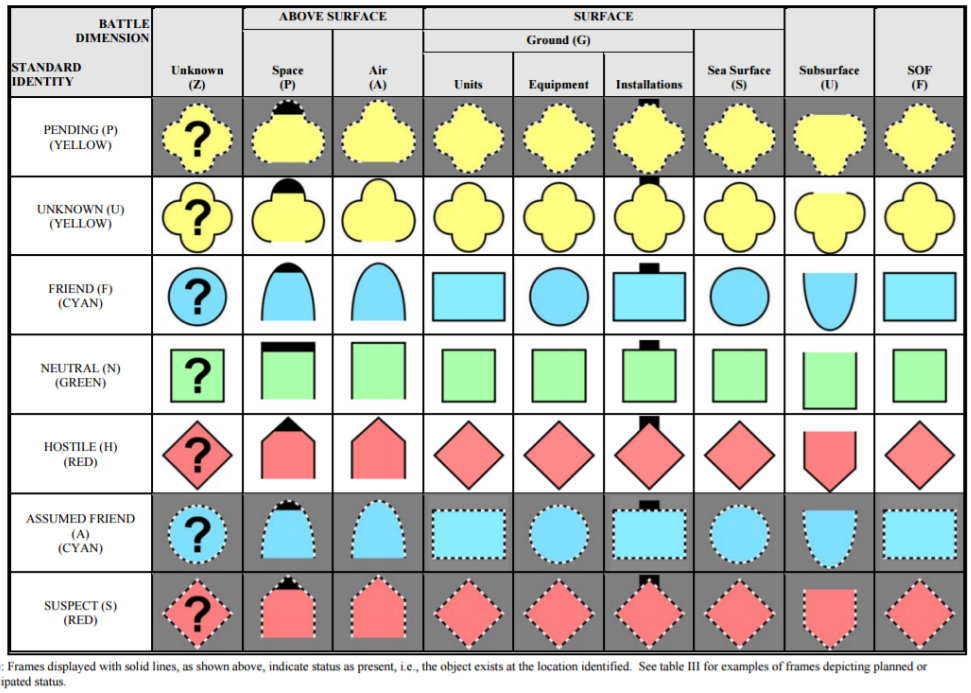



**Figure 2.19:** 2525C Symbology.[28]

Another feature of *Tactics* is that it runs in a web browser (it is web-based) and as such is largely flexible as it runs in different systems, as long as they can execute a web browser.

[28]Source: MIL-STD-2525C, a Department of Defense Interface Standard documents, which defines common warfighting symbology

***Sword*** is a simulator developed by the Masa Group and it was designed to train military commanders or to prepare civilians forces (civil protection) for emergencies.

Because it used the input of 17 different countries, *Sword* has an AI which acts upon a detailed military doctrine, providing an accurate simulation of armed forces, allowing for commanders (scaling from battalion to higher) to train for their positions using *Sword*, giving orders which the AI will know how to interpret and fulfill in a realistic way. The orders given to the units are written via C-BML [29].

The scenarios are created by a *Sword*-specific tool. That tool is fed with terrain information (see 2.6 for more information) and results in a scenario where the user can place units or other entities, like buildings. The units themselves can be edited to better accommodate the user's requirements. After a play session is over, the actions taken can be studied in the After-Action review.

*Sword* allows for the timescale to be changed and while this is not a realistic feature, it allows for trainees to more attentively process what is happening (slowing down) or skipping when nothing is happening in the battlefield. The lines of action where the units are committed appear as a line on the map. It has a 2D perspective, as it can be seen in Figure 2.20, and the units are represented by using the same military symbology as before (visible in Figure 2.19).

In terms of multiplayer architecture, *Sword* uses a client-server approach (dedicated server), where the players can be organized in teams. The players themselves can join the game as:

- Platoon commander;
- Company commander;
- Supervisor (Umpire);

There are only those types of commander because the units themselves are either a platoon or a company.

The different human players taking part in the simulation can communicate using an internal radio system, which can simulate the signal degradation that impacts the communication.

*Masa Sword* also supports integration with other simulation systems by being compliant with the HLA standard. It also allows for the possibility of remote connection to create a distributed simulation.

***MÄK Combat Staff Training*** (CST) is developed by VT MÄK and can provide training for different levels of command (squad leader to brigade). It is designed to work in different conditions:

- Regular warfare;
- Irregular warfare;
- Border protection;

[29] BML is an unambiguous language used to simplify the communication between forces who are conducting military operations, which facilitates the situational awareness and a having a shared, common operational picture.[13]

[30] Source: http://1.bp.blogspot.com/-2k8ESW0F8XI/VUDLQSUg3OI/AAAAAAAAM8U/eyxCoq7alnM/s1600/SWORD%2B3.png

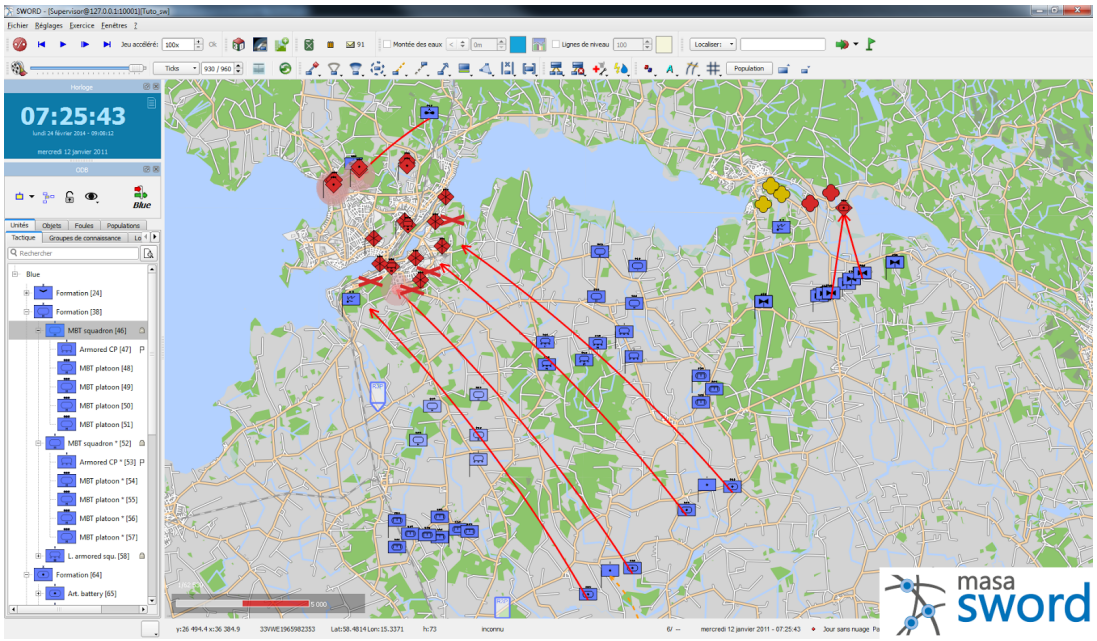



**Figure 2.20:** Sword Interface. [30]

- Civil emergency response.

To build a constructive simulator (CST), VT MÄK purposes combining *VR-Forces*, *VR-Engage* and *VR-Link* in order to build a solution alike the previous ones. Through *VR-Forces*, the training scenarios are created. In order to provide a flexible simulation they are based on terrain databases (allowing to simulate situations on real locations) and unit databases (allowing to use real soldiers and equipment in training). When creating scenarios, it is possible to switch between 2D and 3D for both seeing the overall state of the scenario and for precise placement of units in the map. *VR-Engage* connects humans to the simulated environment and *VR-Link* integrates HLA or DIS standards into the solution.

Through *VR-Engage*, any number of human players can connect to the scenario, occupying one of the roles stipulated to be filled by a human in the scenario. The company was contacted about the roles that may be occupied, but the only roles that have relevance for this paper are commander and instructor. In terms of architecture, one of the client is a host and plays as an instructor. Any number of players can connect, so long as the architecture is able to support it.

*VR-Link* provides a communication's system, with integrated voice or chat communication which can look like different technologies [31] and configured individually in order to provide unique behaviour and features by channel or technology. The communication is established via HLA/DIS.

The camera's perspective varies depending on the user's needs, and in this solution, it is in 2D, where the map (visible in Figure 2.21) shows the relief. The screen has a grid overlay over the map, and

[31] Simulating *UHF*, *VHF*, *HF*, *SATCOM*, *Intercoms*, *Hotlines*, Call answer lines or broadcast nets
[31] Source: https://player.vimeo.com/video/126496197 (adapted)

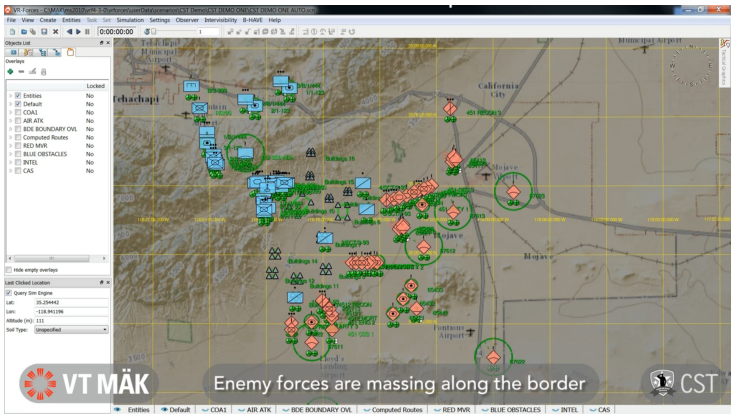



**Figure 2.21:** VT MÄK screenshot.

it allows for arrows to be drawn over it for the courses of action to be viewed.

The AI present in the game comes with different behaviours already scripted, like attacking or patrolling, and allows for creating new scripts in order to accommodate different requirements. The behaviours are composed of tasks. It is also possible to configure the SOP through triggers, to create dynamic plans.

When using the simulator, it is possible to save the current point of the simulation, which allows to replay the scenario from that point. When the session ends, it offers an after-action report.

This simulator executes in a browser to reduce deployment costs.

## 2.9 Conclusions

Given all the information from the previous sections, where we discussed the requirements of a military simulator and analyzed the ones available on the market, it can be concluded that modern simulators have in common the following characteristics:

- Real-time action with timescale manipulation;
- Client-Server architecture with the server making the calculations and the clients being lightweight;
- Implicit line-of-sight;
- Different roles for clients;
- Scenario creation tool;

There are also some common characteristics between the military simulators in the market:

- 2D view of the map;

- Unit symbology in accordance to a standard;
- Unit actions from military doctrine;
- Internal communication systems;
- Implementation of the HLA or DIS protocols;
- Configuration of the SOP;

These characteristics should be implemented in any modern simulator as their existence in some of these simulators makes them important to create a simulator which can be relevant for modern military training.

Various characteristics in the simulator were outlined in a table available at Appendix B.

# 3

# Solution Architecture

## Contents

The following chapter 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 Chapter, which will allow for the proposed system to provide realistic behavior, in the context of this thesis.

## 3.1 Solution Modules

Since the proposed system has a certain degree of complexity, it was divided into different modules (as seen in Fig. 3.5):

- 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 (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:

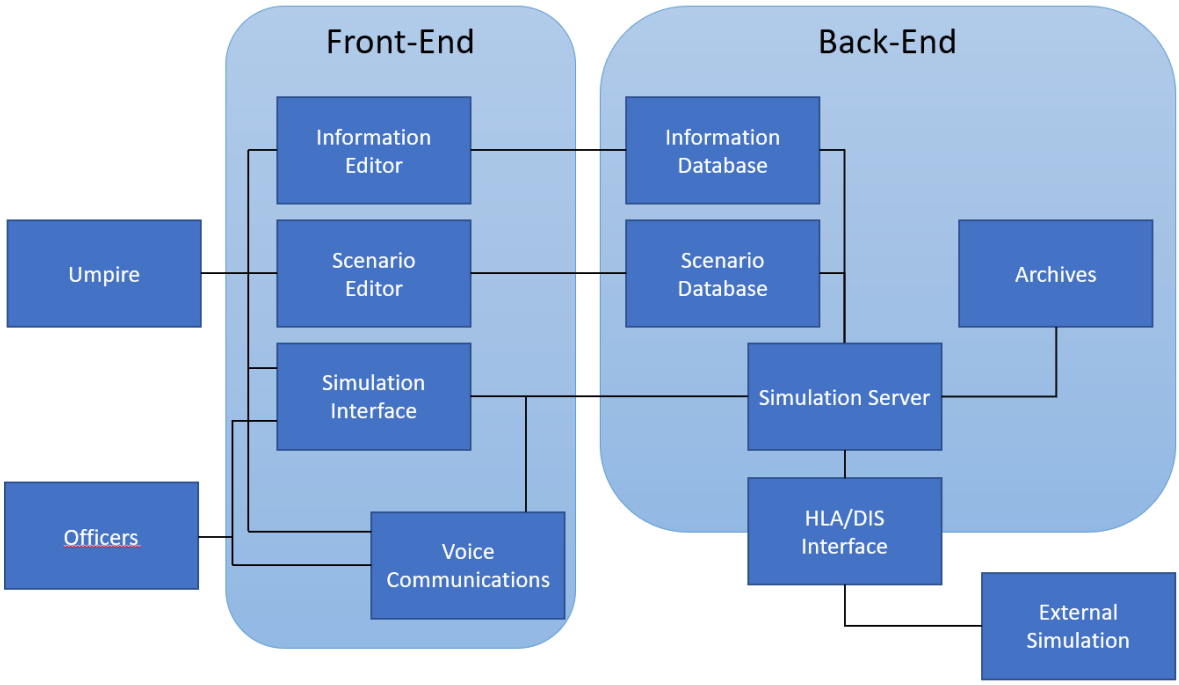



**Figure 3.1:** Internal modules of the Simulator and their relationships.

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;
- 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 3.2.

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 3.3, will issue orders to all its direct subunits, the three platoons.

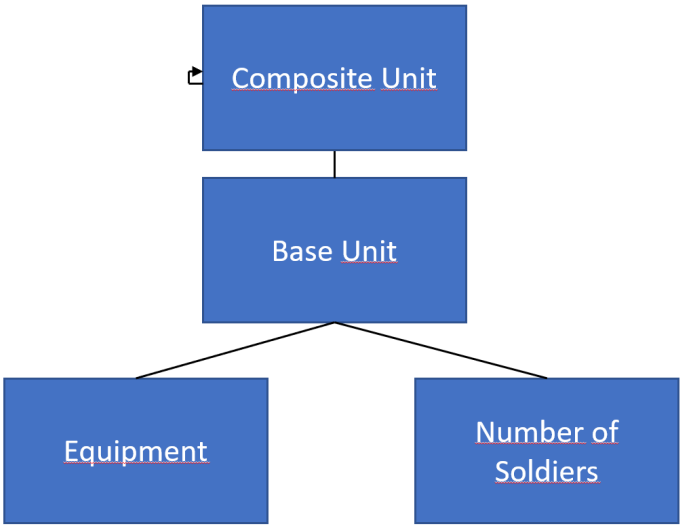



**Figure 3.2:** Internal constitution of a TOE instance

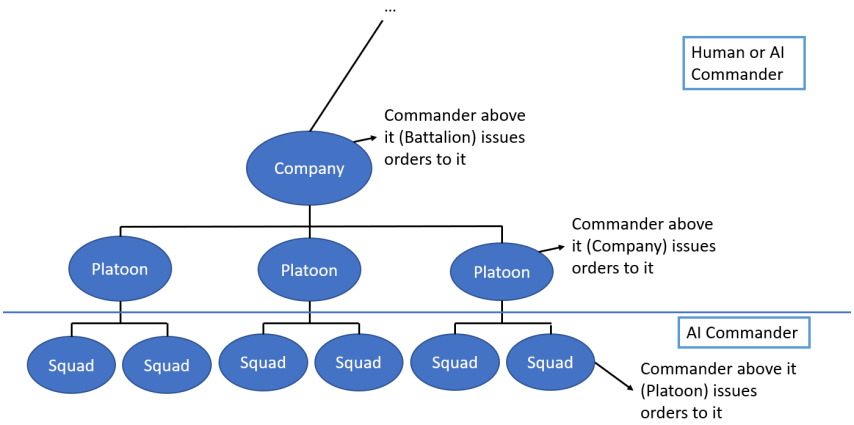



**Figure 3.3:** Hierarchy and commanders in the simulator

**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 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 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 [1], 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

[1] 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.

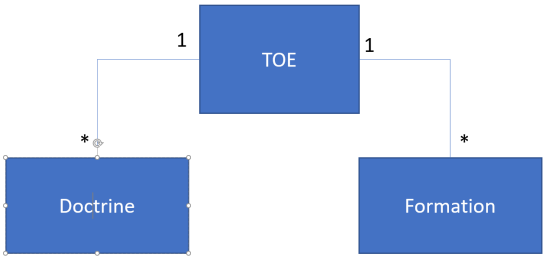



**Figure 3.4:** Relations between the concepts of TOE, Doctrines and Formations

initial parameters (if there are any) are mapped to the parameters of the orders below according to what was defined in the doctrine.

Having described these three artifacts, TOE, Doctrines and Formations, it is shown that they are related in the manner pictured in 3.4.

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.

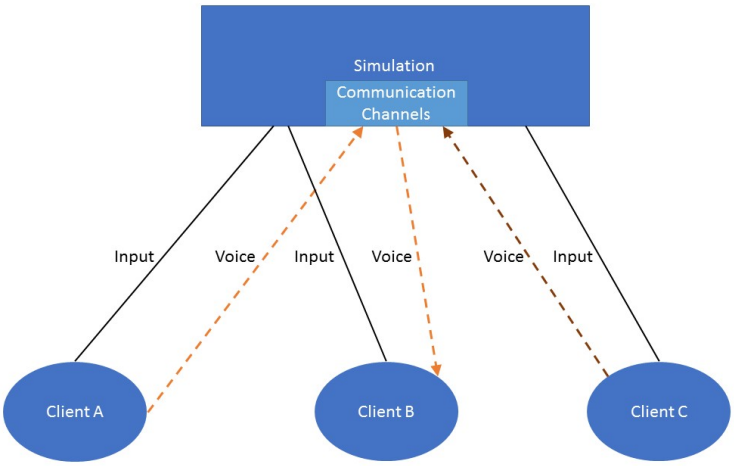



**Figure 3.5:** Example of multiple clients connecting to the simulator. Each color of the dashed line represents a different channel. When client A sends a communication, client B that is on the same channel receives that communication. Client C is in another channel, so not only it does not receive the first communication, as no client receives C's communication

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

# 4

# Solution Implementation

## Contents

This chapter describes the developed POC in order to demonstrate the feasibility of the model described on the previous chapter.

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.

## 4.1 Unreal Engine Architecture

Given the characteristics of this project and the complexity involved in creating a RTS (which is the type closest to the simulator's characteristics), in order to develop this project as quickly as necessary, as discussed before, *Unreal Engine* blueprint's were used.

As *Unreal Engine* is a complex system to use, it is necessary to understand it's architecture in order to use it. Unreal Engine comes with various predefined classes each of which has a specific role during gameplay. In blueprints, the most important classes to understand are:

- Game Mode
- Game State;
- PlayerState;

- GameInstance;
- Actors;
- Controller;
- Widgets;

***Game mode*** contains all the rules pertaining to a game like the number of players, the steps taken when a player joins a game and the overall gameplay flow. The latter is managed through different exposed events which pertain to various game important events. Each map has a game mode associated with it and it is possible to use different game modes in different maps. Game mode controls how the game flows and thus, it only exists on the server.

***Game state*** contains the information about the game that is important for all the players connected to a game and it can be used to establish a way for the different clients to communicate between them.

***Player State*** contains the information pertaining to a individual player, like its score, number of deaths or other information.

A ***GameInstance*** instance is created for each connecting player. It is usually used to transfer information between the different maps, as during their transition, all other instances of other objects are destroyed.

***Actors*** are all the entities in the game world. A special type of actor which is very important to mention is the Pawn. They are used to represent a will (either human or AI) during a session. As actors exist in the world, they interact with the world in terms of collision and other physical interactions. It also determines the visual representation of a player and its location within the game.

The ***Controller*** is the class which allows for a player to transmit its will into the possessed pawn. There are two different versions of the Controller class: the *PlayerController* and *AIController*. As the names suggest, the *playerController* is used when a human connects and an *AIController* is used by an AI. The Controllers and the pawns have a one-to-one relation, which means that a controller can only possess a single actor at a given time.

***Widgets*** are classes specially made for creating UI elements. Through widgets, it is possible to create all the 2D or 3D interfaces like the game's menus or the player HUD.

## 4.2 Proof of Concept

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 3.1, namely the Simulation Interface, the Simulation Server and the Unit Database.

The Simulation Interface module is mainly constructed via widgets witch allow for the various UI elements to be added to the screen during the session. This module also contains the creation of other UI elements like the fire lines and movement lines, as shown in Figure 4.2.

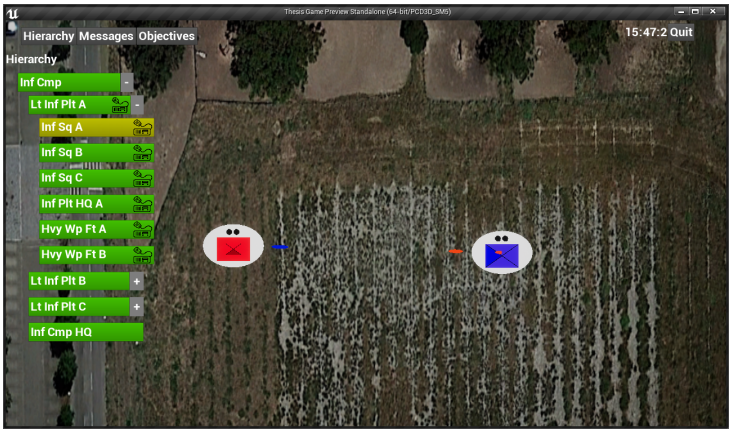



**Figure 4.1:** Visual representation of units firing during the simulation. The blue line represents friendly fire and the red line enemy fire.

In figure 4.1 it is possible to see two lines (one red and one blue, representing enemy and friendly fire, respectively). During the simulation, these lines will move from the firing unit to the targeted unit.

A feature which is implemented in this module is the *order mode*. It is a system that allows the commander to give orders to the units which are under its command.

The system uses an actor, a *waypoint node*, to save each order which is created during the *order mode*. After the commander exits the *order mode*, all orders whose parameters were fully attributed are issued to the unit.

The *order mode* is started by a button in the interface, when selecting a unit which is controlled by the commander. By clicking that button, the system contacts the server to get all the actions that this commander can issue, according to the faction's doctrine.

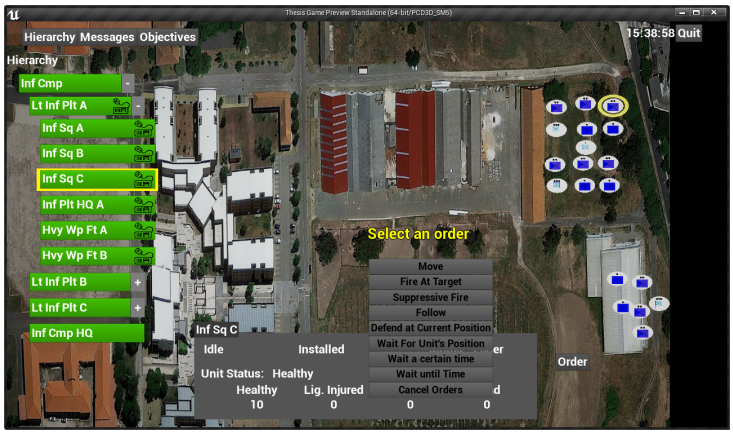
After getting all the actions' names, it generates a list of buttons, one for each order. When an order is selected, a *waypoint node* is spawned and will cycle through all the parameters that the order requires in order to be fulfilled.

Subsequently attributing all the parameters with the value, the system will either ask for the next order, if it was set for allowing orders after it, or it will end the order mode, transferring all orders given to the server, which will then begin their decomposition through the doctrine as described in Section 4.2.5.

The various steps to give an order in the POC are shown in image 4.2.


**(a)** Order button


**(b)** Choosing a target for the orders




**(c)** Actions available for the an infantry squad

**Figure 4.2:** UI process to issue orders to units in the POC. [1]

### 4.2.1 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.

Next a list of the most important classes of which the POC is composed of are presented:

- Master Unit - *Pawns* which represent the military units present on the map;
- Master Commander - *Pawns* to be possessed by each player;
- Master Terrain - *Actors* placed on the map to represent the different types of terrain;
- Master Weapon - *Actors* to represent the weapons that can be employed by the units;
- Doctrine - Class that allows to separate unit behavior and orders from the actual implementation of the units and so, facilitates the creation and addition of new doctrines to the simulator.
- TOE - Represents the implementation of a TOE and it allows to separate the actual unit internal information from the unit implementation, allowing for more flexibility for the Master Unit class.

Along with theses specific classes, some other *Unreal Engine* specific ones were developed:

- Simulator Mode;
- HumanController;
- UnitBrain;

The ***Simulator Mode*** is a GameMode used during the simulations, loading the required information from XML files (ammo, weapons, units' characteristics, including their corresponding TOE and Doctrines, and respective positions) and then managing unit and commander spawning at the beginning of the game. During the session, it will also provides calculations for the various aspects of the simulation, like the visibility mechanics, the influence map updates and the orders given during the simulation. Because the GameMode only exists on the server, some precautions must be taken before issuing a call to the GameMode.

The ***Human Controller*** and the ***Unit Brain*** are the *Controllers* which are used to allow the users (either human or controlled by an AI) to interact with the virtual world. Via a process called possessing either *commander* pawns or the *units* during the simulation. Controllers exist in both the clients and the server, making it the best choice for the clients to make requests for the server.

***Master Unit*** defines all the common functionalities which are reused to all the units, like the detection system and the execution of the various types of basic orders. From it, subclasses were created (*MasterCombat* and *MasterHQ*) in order to define two units which have different default behavior: a combat unit should engage when it sees an enemy unit while a support unit, like a headquarters, should not engage enemy units. As the prototype only has one type of unit (Infantry) and one type of support unit (the headquarters), no more subclasses were created.

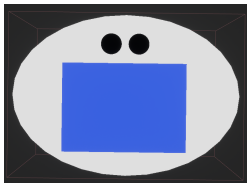


**Figure 4.3:** Generic representation of a unit.

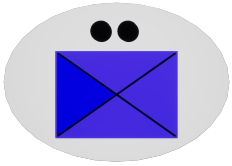


**Figure 4.4:** Representation of an friendly infantry squad.

**Figure 4.5:** Representation of an enemy infantry squad.

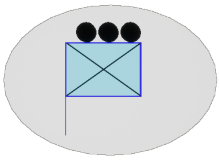


**Figure 4.6:** Representation of an infantry platoon headquarters.

***Master Commander*** defines the basic actions of all commanders, in terms of unit selection and camera control. Its child classes are used to represent the two roles that a player can have: a commander of a formation or the umpire of the session. The different kinds of orders are determined through the Doctrine associated with the commander's faction.

***Master Terrain*** contains two components: a *Nav Modifier Component* and an invisible cube. Using this class has two different effects:

- The *Nav Modifier Component* changes 4.7 the *Nav Mesh*[2] in terms of movement cost within the map so it represents the difficulties or advantages of transposing the different type of terrains.

- The invisible cube is used to represent the terrain's height and it is configured to block the traces used during the visibility calculations in order to simulate 3D visibility in a 2D representation.

Before creating the concrete terrain classes, two more abstract classes were created, ***Revealing Terrain*** and ***Concealing Terrain***. These classes have to do with visibility calculations and generally represent terrains that provide cover from enemy units.

In the coming subsections, the more complex algorithms that are part of the calculations made by the simulation server are described.

[2]Volume used by AI systems that allows them to be able to find a path in the game environment

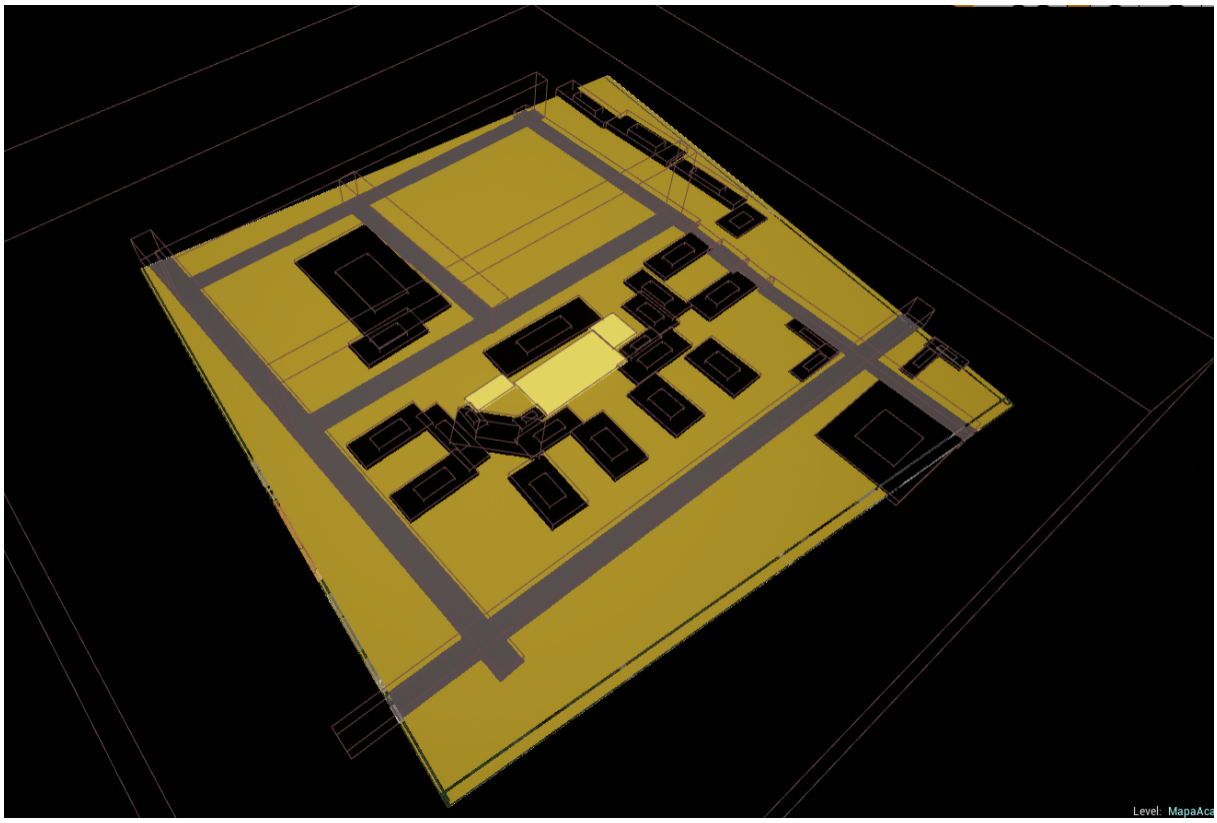



**Figure 4.7:** Navigation Mesh of the demonstration map in the POC. Each color represents a different type of terrain, where gray are the roads, black are the building, yellow is normal terrain and light yellow are zones restricted to infantry units.

### 4.2.2 Arranging a Session

When a session is created, the player that created the game (server) chooses the scenario to use and assigns himself and the rest of the connected players the different roles that they will play on the coming session.

The possible roles are:

- Umpire (One per session) - can control any unit controlled by an AI and can pause the simulation. There can be only one umpire (if any) per session;
- Commander - issues orders to a given formation;

After all the players are assigned positions and the server player starts the session, the server opens in the map associated with the session, loads and spawns the units and then players are placed in their assigned roles. When all the players have fully loaded, the session starts.

### 4.2.3 Visibility Calculations

In a real life situation, whenever units are sighted during combat they are reported to the mission command on the battlefield, so it can take appropriate action to the developments. In the developed simulator and particularly the POC, the *Master Unit* instances are the ones that sense and report their units location.

This is done by using one of *Unreal Engine* 's pre-existing systems, the *AIPerception*. *AIPerception* will fire an event whenever it detects a actor on the sensing radius defined for that actor.

There are two parts of the visibility calculations, the UI representation and the internal information of each unit (their individual knowledge about the environment). The UI representation will only update when the player is not playing the role of an umpire, as in that case, they are always visible in the UI. The internal state of the units is always updated. It is checked whenever there are behaviors which involve an enemy unit, like when engaging a unit (if it is still available). Regardless of the commander's type, as the internal state of the unit must be updated the following algorithm is executed:

**Algorithm 4.1:** Visibility calculations algorithm

```
Data: Detected enemy's location
Result: Check if the unit is visible to the sensing unit
if the sighted unit is in combat then
    return true;
else
    TerrainsToIgnore = Terrains that the unit is in the border of and that the seen unit is also in;
    forall Concealing terrains between the unit and the seen unit, except the Terrains to ignore)
    do
        if There is a concealing terrain in the path then
            if The seen unit and the unit are both on that terrain then
                if The seen unit is on the part that provides cover then
                    return false;
                else
                    if There is path between the units that does not pass on the covering part of the
                      concealing terrains then
                        return true;
                    else
                        return false;
                return false;
            else
                return false;
    return true;
```

Regardless of the final result about the sensed unit visibility, it will be broadcast to all the non-umpire commanders in the same faction through the *Controllers*. This visibility algorithm works for the POC units, but if there were other kinds of units (camouflaged units) in the simulator or if we had to taken into account modifiers to the visibility, like weather for instance, this algorithm would have to be modified.

Another feature that is associated with the visibility calculations is the influence. Whenever a unit is sighted, the influence map will modify the *nav mesh* so that units, unless expressly commanded, will try

to avoid the locations of the sighted enemy units. The heatmap is created through the grid created on the map, enabling or disabling a *Nav Modifier Component* to influence the *nav mesh*.

### 4.2.4 Fire Mechanics

When a unit receives a direct fire order, the algorithm which is followed is as follows:

**Algorithm 4.2:** Fire Order algorithm

```
Data: Direct fire order is issued
if the designed target is a valid target then
    if the target is within range? then
        EngageTarget();
    else
        ApproximateToTarget();
else
    if Are there other targets in range then
        ChooseBestTarget()
    else
        if had a target and it should pursue it then
            MoveToTargetLastLocation()
        else
            AwaitFurtherOrders()
```

A part of the algorithm that will be further detailed is the engagement of units:

**Algorithm 4.3:** Engage target function

```
Data: Fire order is issued
if the unit is ready to fire then
    CalculateModiers() ;
    CalculateNumberOfBullets() ;
    SetCooldownToFireAgain() ;
```

The modifiers are of two types, offensive and defensive. In terms of offensive modifiers, the simulator takes into account:

- Unit's current morale and status;
- Number of capable soldiers inside the unit;
- Types of terrains that were crossed;
- Distance to the target (depends on the units that were used);
- The experience of the unit;

For defensive multipliers, the simulator considers the following:

- The target's resistance to the caliber discharged against it;
- The target's status and morale;

The offensive and defensive multipliers have the following mathematical relationship:

$$\frac{\prod OffensiveMultipliers}{\sum DefensiveMultipliers} = damageDoneToTheTarget \tag{4.1}$$

When a unit takes damage, it follows this algorithm:

1. Select a random soldier belonging to the unit to take the damage;
2. Check if there is need to change the unit's status;
3. Switch the unit to the pinned status, diminishing the unit's speed and enhancing its defense;

The suppressive fire algorithm is similar but instead of receiving enemy units as an input, it receives positions. It affects a area instead of a single target and so it has the potential several units at once. However, its effectiveness is reduced when compared with the direct fire.

### 4.2.5 Doctrine

As stated in Chapter 3, 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.

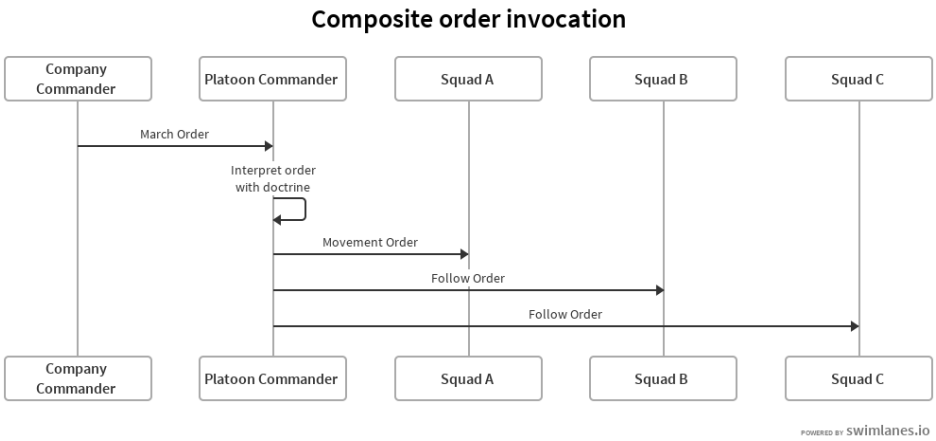



**Figure 4.8:** Invocation of an composite order

In figure 4.8 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.

The last module that will be described will be the Simulation Database, describing both how the information is stored and how is it modified by a user.

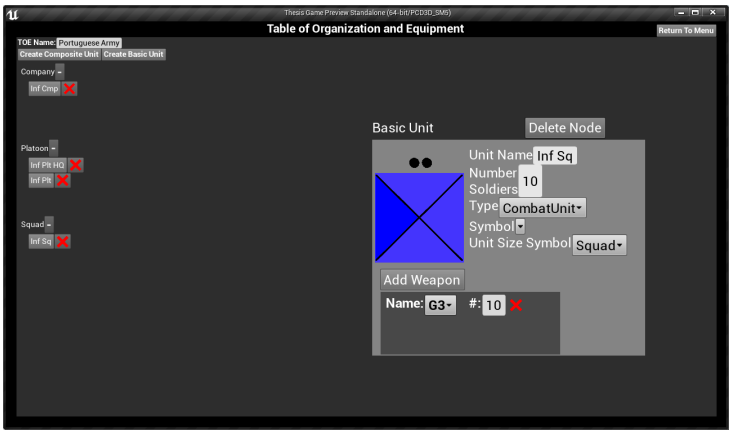



**Figure 4.9:** Interface that allows a user to manage a TOE, showing a basic unit.

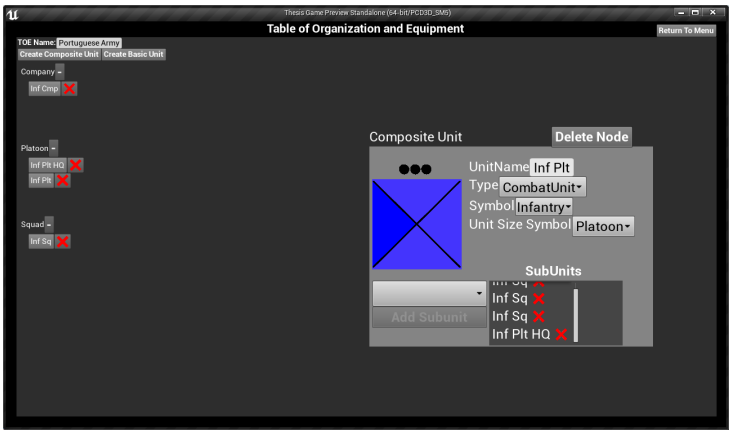



**Figure 4.10:** Interface that allows a user to manage a TOE, showing a composite unit.

### 4.2.6 Simulation Database Implementation

The purpose of this module is to store information that is relevant for the simulation, such as *Doctrines*, TOE, Formations and Weapons.

For the POC, two different editors were developed: the TOE, Doctrines and Formations' editor and the Scenario Editor. As the names suggest, the first editor is used to create and edit TOE, Doctrines and Formations and the second one uses the artifacts created by the first to prepare situations which can be used for the training of officers. The first editor can be divided into three different modules, where each one creating a different artifact. In the first module, a TOE is first created by assigning it a unique designation. The user should then create the basic and composite nodes, whose internal structure is described in 3.1, until it describes all the units that should be detailed in this TOE. In this implementation, a single TOE describes multiple units at once.

In the figures displayed at 4.10, it is possible to see a TOE's edition screen, where it is possible to

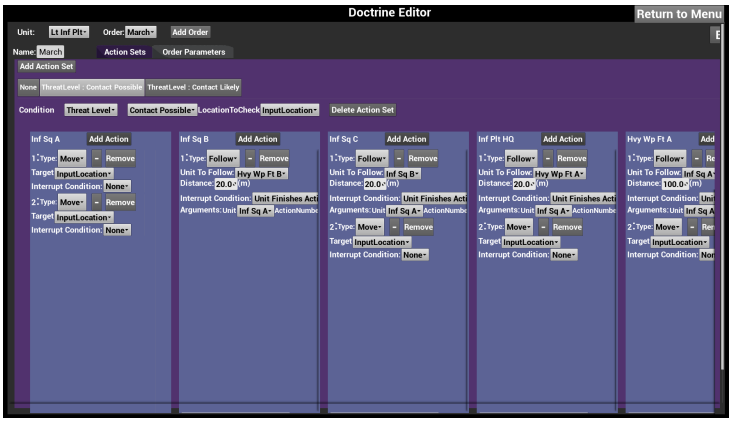



**Figure 4.11:** Interface to manage how a composite order is executed.

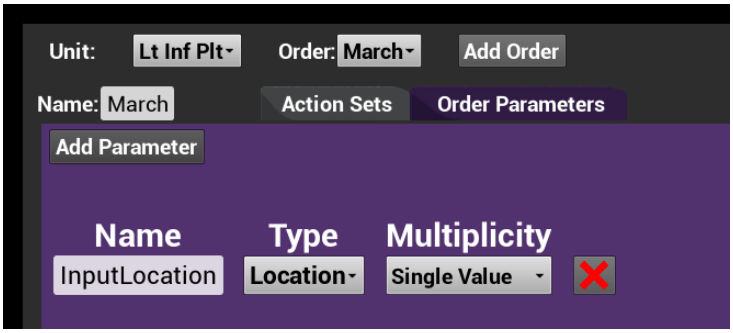



**Figure 4.12:** Interface to manage a order's parameters.

assign a name to the TOE and add new basic and composite units. As new units are created, they are organized on the list on the left side of the image, where they are ordered decreasingly from the units with the highest level of hierarchy. After a unit changes its Size Symbol, it is placed in the appropriate category. In the first image, it is possible to see a basic unit. These units will form the base from which the other units, the composite units, are made of internally. The displayed example, an infantry squad, is constituted by ten soldiers who use a G3 rifle. The second image shows a composite unit, an infantry platoon, which is made from those infantry squads.

The second module is responsible by the creation of Doctrines for use in the simulator. As in the first module, the process begins by creating a Doctrine and assigning it a unique name. After that, the various orders associated with the units who can be assigned custom orders are created via the interface displayed at 4.12 In this interface, it is possible the various elements which are contained in a doctrine, as described in Section 3.1, like the order's parameters (visible in image 4.12) and its action sets. As more action sets are created, new buttons are shown, where the action set's condition for execution is visible. Inside each action set, it is possible to attach actions to any of the subunits that were defined in the previous module and to create an interrupt condition for that action.

Finally, the third module is used to create formations. It begins by assigning it a unique name and a TOE. After selecting a TOE, it will be possible to add instances of the TOE nodes that were created by the first module. After adding an instance, the interface allows for the user to change its unit's names. In the image 4.13 it is possible to see an example of the formation editor.

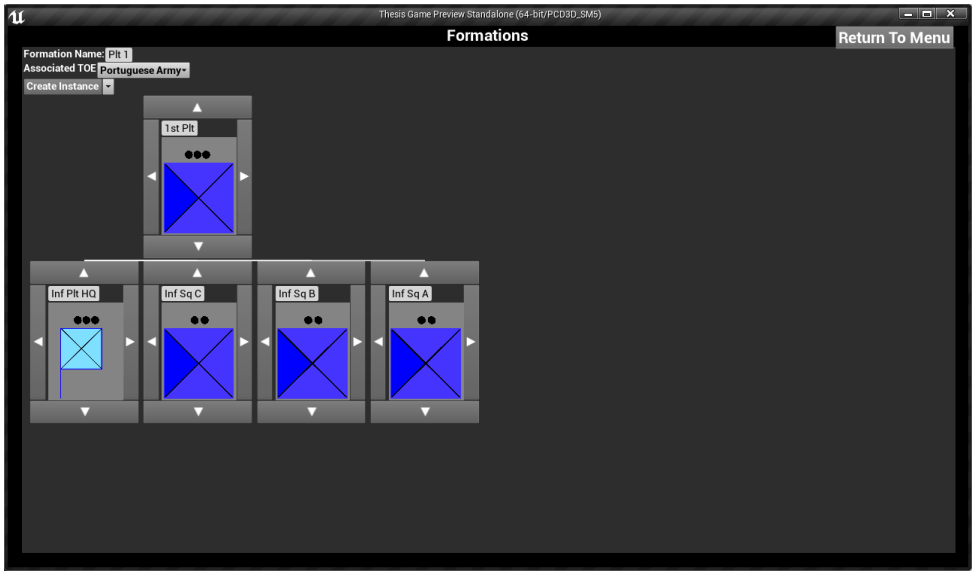



**Figure 4.13:** Interface to oversee a formation's edition.

At the beginning of a session, the Simulation Mode class loads the relevant information, starting with the session's XML, which contain the session's factions. As a faction is constructed from a TOE, a Doctrine and one or more formations, the Simulation Mode will then load the appropriate files, only loading the ones that are required instead of the entire XML database.

The other editor, the scenario editor, is used by the user to create and prepare scenarios to use during the training sessions. Creating a scenario has two separate phases: first a name and a map for the scenario must chosen and after which the chosen map and the user can set the forces that will be present at the start of the session.

It is only possible to have two factions in a session in the POC. However, it is possible to change their name, TOE and doctrine, as displayed in the figure 4.14. It is also not possible to customize the color associated with a faction when editing a scenario (as during the training session, the unit's colors will depend on their relations with the commander's faction).

After setting a faction's details, the user can add any formation associated with the faction's TOE to that faction, as long as that formation was not already added to another faction that uses the same TOE. In a first phase, the units are placed in an unplaced list and as they are placed in the scenario, they are transfered to another list which displays the units associated with a faction. In the figure 4.15, all the units have already been placed.


**Figure 4.14:** Interface to edit a scenario's factions.

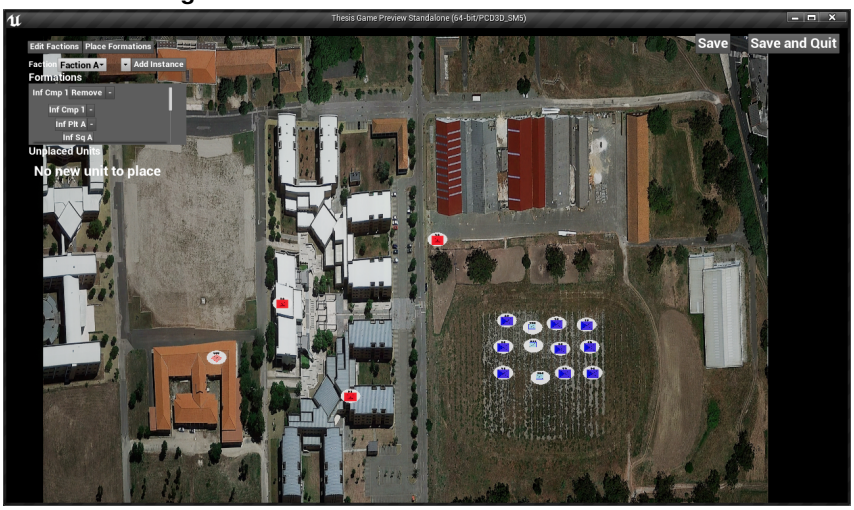



**Figure 4.15:** Interface to manage the formations added to each faction.


**Figure 4.16:** Interface to assign an initial order to a unit.

In the unplaced units list, it is only possible to select the unit's that are actually displayed during a session (the basic units) while on the other list, we can select any unit in order to see its details and set initial orders during the game. When selecting the unit's headquarters, it is also possible to issue orders

for the whole unit and set generic textual objectives to be displayed to the human commander which will control this unit. When a unit is selected, the displayed interface is the one visible on 4.16, showing the full hierarchy of that unit, its name and showing a button to attribute an initial order to that unit.

The orders that a unit can be assigned in the scenario editor will depend on the doctrine that was selected for the faction. The interface to give the orders is the same that is used during the sessions.

In the next Chapter, we will discuss what was created in order to test if the developed POC showed that the model proposed in this thesis can define a realistic military simulator with a lower cost that those available on the market.

# 5

# Validation

## Contents

In this Chapter, 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 proposed 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.

## 5.1 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.

#### 5.1.0.A System Usability Scale

The System Usability Scale (SUS) 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 [28].

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 [27].

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 [27].

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.

## 5.2 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.

### 5.2.1 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 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 squads of one 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 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 games on them, 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 our testers did not have that knowledge, the results of the tests were affected negatively. According to Jakob Nielsen, a specialist in computer-

human interaction, there should be 5 testers in the these usability tests [21]. In our case, we had 6 people perform our tests. As stated before, in the first phase the testers had to use the Information Editor to create information for the next phases. The complete results of the various questionnaires can be viewed on Appendix C. At the end of each phase, the testers performed the SUS test to that module. For the first tested module, the Information Editor, the SUS results were the following:

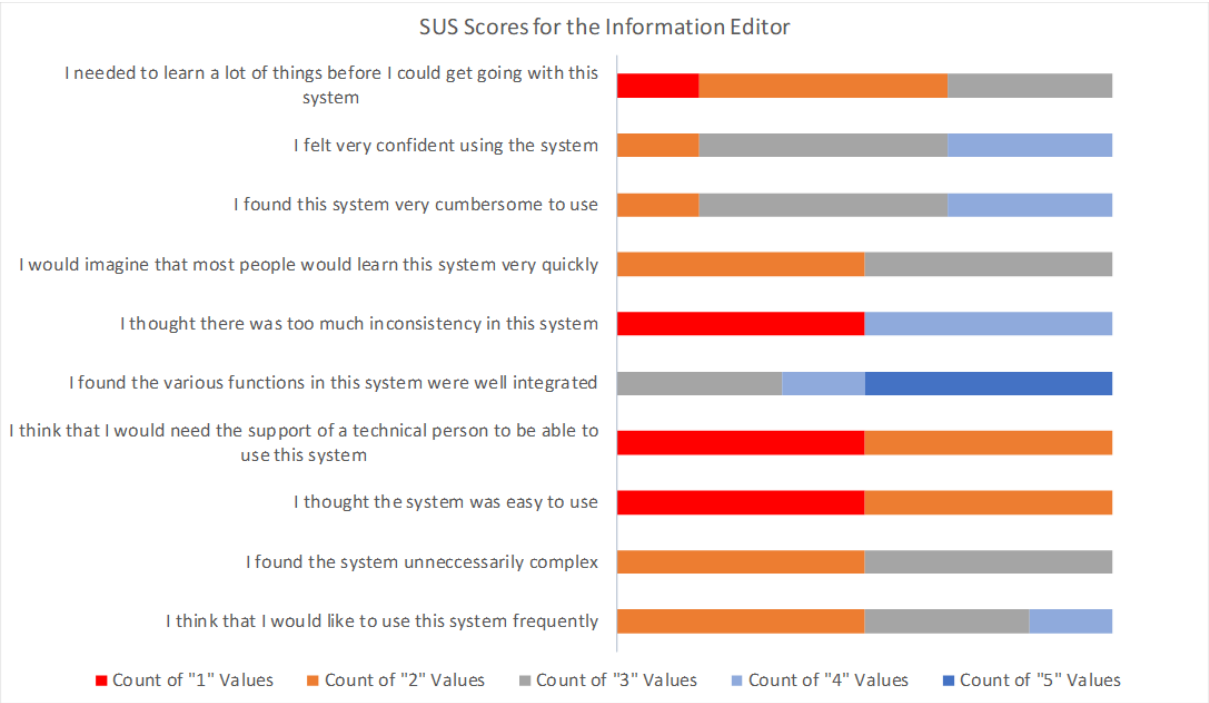



**Figure 5.1:** Result for the SUS for the Information Editor

Calculating the SUS score as indicated in 5.1, its value will be 55 in a scale from 0 to 100.

After constructing their own TOE, Doctrine and Formations, the testers used the Scenario Editor to prepare a scenario using that information. The results of the questionnaires performed after these section of the tests are available on Appendix C. Similarly to the last questionnaire, a SUS test was performed and whose results are the following:

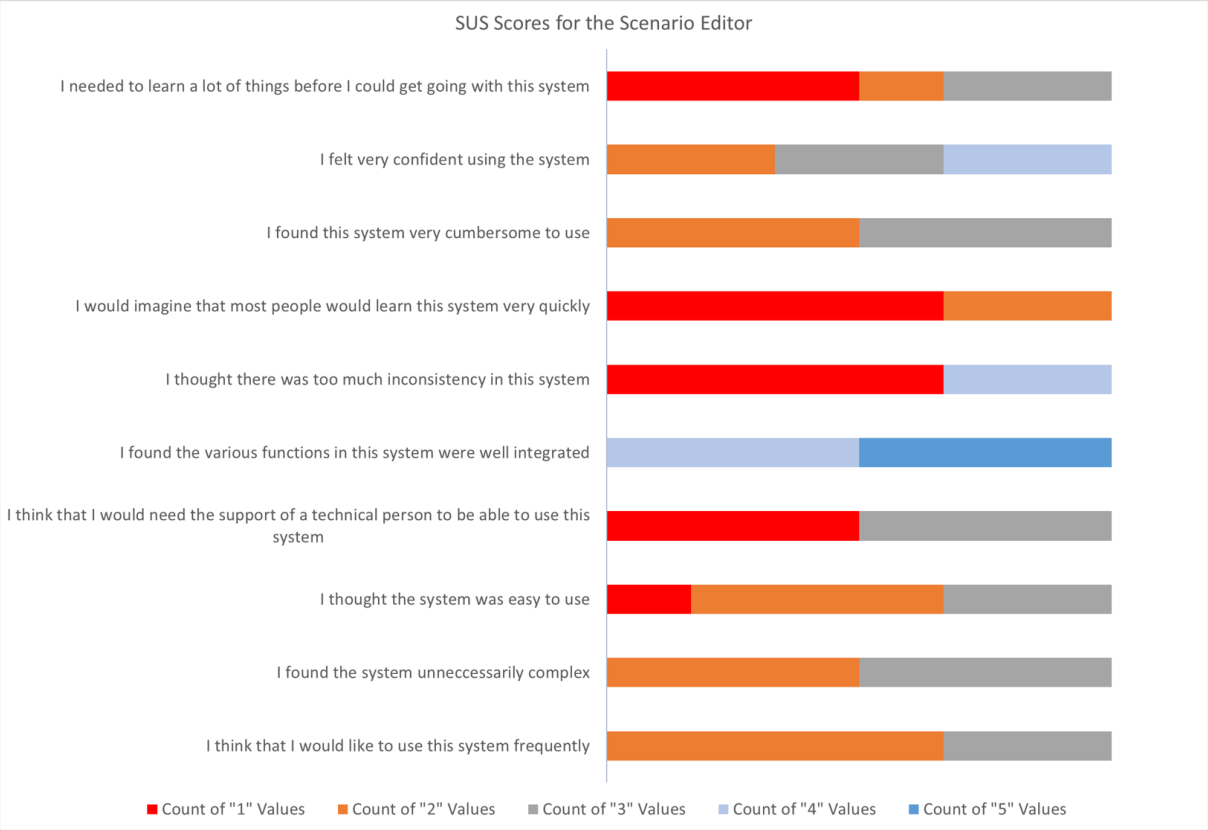



**Figure 5.2:** Results of the SUS for the Scenario Editor

By following the algorithm to calculate the SUS score as indicated on 5.1, its value for the Scenario Editor was 52 in a scale from 0 to 100.

Finally, after finishing the scenario and assigning an initial order to a unit, the testers would create a simulation using that scenario to evaluate the simulation's interface. The results of the questionnaire are available on Appendix C.As before, a SUS test was made and the results are displayed in Figure 5.3. By calculating the SUS score as indicated in 5.1, its value for the Simulation was 77 in a scale from 0-100.

Analyzing the results given by the various questionnaires and the SUS, it can concluded that the interface of the system is one of its major weak points, as it was never the focus of the development. More specifically, both of the editors interface's are the worse ones while the Simulation has the best values. This results can be justified by the fact that both editors require some military knowledge to understand the flow and the reason for the various elements in the interface.

The lack of emphasis on the interface development resulted in a POC which, despite implementing

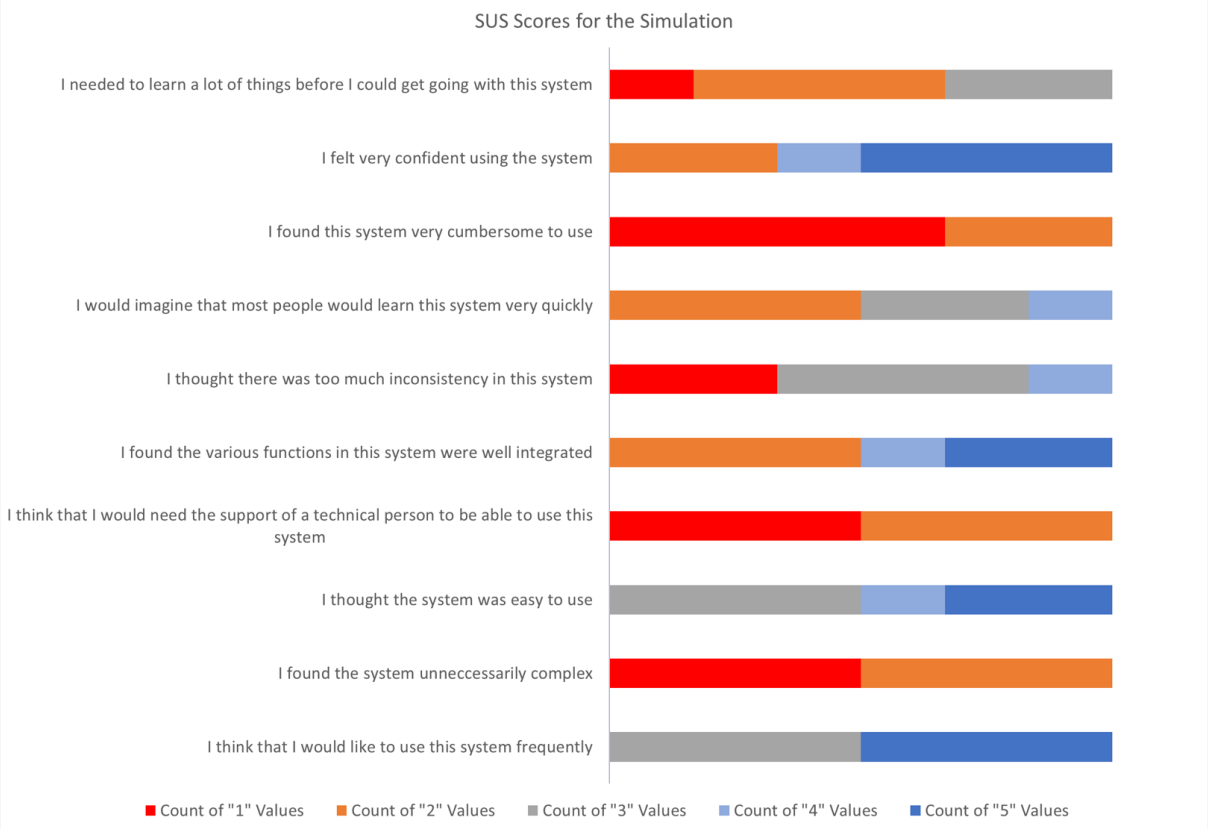



**Figure 5.3:** Results of the SUS for the Simulation

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.

However, as the next set of tests show, this evaluation was influenced by the fact that the testers did not have the military knowledge to understand what they were doing and the flow of the interface.

### 5.2.2 Second set of tests

The second set of tests were dedicated to the evaluation of the realism of the developed prototype. As the POC is of a military nature, these set of tests were only done by people with knowledge of military tactics and behavior. More specifically, the tests were planned to be done 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 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 with the tests, as they were numerous, we would collect their statements regarding the simulator in general, emphasizing the best and worse 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 its features.

More specifically, there were three officers who came from the Military Academy: Tenente-Coronel 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.

During the setup for the training simulation, the POC had some problems and the tests could not be performed as designed. Instead, the officers used the POC individually in order to access the its state. More specifically, it was not possible to connect the three officers to the simulation.

After finishing the tests and the various officers taken part in the exercise and demonstrations, the officers were asked about the strong and weak points of the prototype and of the model. Regarding the positive aspects:

- **The program is simple to understand and manipulate** - Through this evaluation, it is demonstrated what was pointed before about how the understanding of military concepts is important to use the program;

- **The model seeks to integrate with other tools in a seamless way** - The officers referred that there were tools created by other departments within the military whose results could not be integrated into their current simulator. As such, as according to our model, the simulator should be able, for example, to use real terrain information and having that terrain information directly translated into the simulated world, represents a major advantage according to the officers. As 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, its integration into the simulator will ease future developments;

- **The unit's AI behavior follows a doctrine, which is very important in a simulator** - When we designed the AI for POC, we strived to make it as accurate as possible, which was noted and appreciated by the military, demonstrating that, when creating realistic behavior for military units, we should try to emulate what is inscribed in a military doctrine.

However, the officers also pointed out some errors on 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.

These results demonstrated that the developed POC was able to exhibit the potential of the proposed model to be used by the military as the projected features were shown to be an improvement over what is currently available for the Portuguese military.

## 5.3 Conclusions

From both of the test sets made we can conclude that while the POC has many interface problems, which should play a bigger part on the development of future versions. However, with results from the tests done with the military, it can be shown that the overall model is well created and it can serve as a basis for a military simulator for the training of army officers.

# 6

# Conclusion

## Contents

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 2, are related to the architecture and units requirements: by creating units through TOE as described in Chapter 4, 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 4. 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.

## 6.1 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.

## 6.2 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.

# Bibliography


[1] R. Baillergeon and J. Sutherland, Tactics 101: 014. decision-making and the power of ccir, http://www.armchairgeneral.com/tactics-101-014-decision-making-and-the-power-of-commanders-critical-information-requirements.htm, Accessed on 15/11/2016, Mar. 2007.

[2] J. Barfoed, "Military strategy vs military doctrine", Krigsvidenskab, [Online]. Available: https://krigsvidenskab.dk/military-strategy-vs-military-doctrine.

[3] M. Caffrey, "Toward a history based doctrine for wargaming", Strategy Page, 2000. [Online]. Available: https://www.strategypage.com/articles/default.asp?target=warghis2.htm.

[4] "Chaturanga", [Online]. Available: http://www.ludoteka.com/chaturanga-en.html.

[5] K. Cisco, Crisis in the caucasus: The russo-georgian war. K & R Games LLC, 2016.

[6] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AI: an introduction", CoRR, vol. abs/1709.00084, 2017. arXiv: 1709.00084. [Online]. Available: http://arxiv.org/abs/1709.00084.

[7] O. G. Consortium. (2012). Ogc® geography markup language (gml) — extended schemas and encoding rules, [Online]. Available: https://portal.opengeospatial.org/files/?artifact_id=46568.

[8] A. Cunha, "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, 2011.

[9] "Dictionary.com unabridged", Apr. 2018. [Online]. Available: http://www.dictionary.com/browse/simulation.

[10] H. Dunga. (2009). Wargames??? yes you can!, [Online]. Available: https://boardgamegeek.com/geeklist/42654/wargames-yes-you-can.

[11] K. E. Foote and M. Lynch. (1995). Geographic information systems as an integrating technology: Context, concepts, and definitions, Department of Geography, University of Texas, [Online]. Available: http://www.colorado.edu/geography/gcraft/notes/intro/intro.html.

[12] R. M. Fujimoto, The high level architecture, http://www.acm-sigsim-mskr.org/Courseware/Fujimoto/Slides/Fujimoto 20-HighLevelArchitectureIntro.pdf, 09/12/2016.

[13] K. Galvin, M. Hieb, A. Tolk, and C. Turnitsa, "Report from the coalition battle management language study group", Naval Postgraduate School, Monterey, report, 2005.

[14] I. L. Holdridge. (). Tac ops 4: Overview. http://www.battlefront.com/products/tacops4/tacops4.html.

[15] A. P. Jackson, "The nature of military doctrine: A decade of study in 1500 words", The Bridge, [Online]. Available: https://www.realcleardefense.com/articles/2017/11/15/the_nature_of_military_doctrine_a_decade_of_study_in_1500_words_112638.html.

[16] M. Land and G. Billingsley, Next war: Korea living rules. GMT Games, LLC, 2012.

[17] W. Lee. (2010). High-level architecture (hla) versus distributed interactive simulation (dis), [Online]. Available: http://stackoverflow.com/questions/3887460/high-level-architecture-hla-versus-distributed-interactive-simulation-dis.

[18] T. Mali, "P0rqu3 3rr4m05?", Revista Galileu, pp. 1–2, 2011. [Online]. Available: http://revistagalileu.globo.com/Revista/Common/0,,EMI221103-17773,00-PRQU+RRM.html.

[19] J. Mattson, "Simulation ammo allows for true-to-life training", Mar. 2012. [Online]. Available: https://www.army.mil/article/75725/simulation_ammo_allows_for_true_to_life_training.

[20] M. Moore, "Arab forces, marines take kuwait city", The Washington Post, 1991. [Online]. Available: http://www.washingtonpost.com/wp-srv/inatl/longterm/iraq/timeline/022891.htm.

[21] J. Nielsen, "How many test users in a usability study?", Nielsen Norman Group, 2012. [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/.

[22] D. Poole, A. Mackworth, and R. Goebel, Computational intelligence: A logical approach. Oxford University Press, New York, 1998.

[23] E. Quelopana, Conhecimento e decisão: Um estudo sobre a relação entre o conhecimento e a qualidade de deci 2003.

[24] S. J. Russell and P. Norvig, Artificial intelligence: A modern approach, 2nd ed. Pearson Education, 2003, ISBN: 0137903952.

[25] J. Santos, A simulação. contributos para a formação e treino, 2012.

[26] C. R. Shrader, History of operations research in the united states army. Department of the Army, 2009, vol. 3, ISBN: 978-0160794254.

[27] N. Thomas. (). 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/.

[28] usability.gov. (). System usability scale (sus). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[29] A. Visacro, “O desafio da transformação”, Military Review, pp. 46–55, Mar. 2011.

[30] M. Watson, Sand table for military purpose, http://pt.slideshare.net/watsonmary/sand-table-for-military-purpose, 2014.

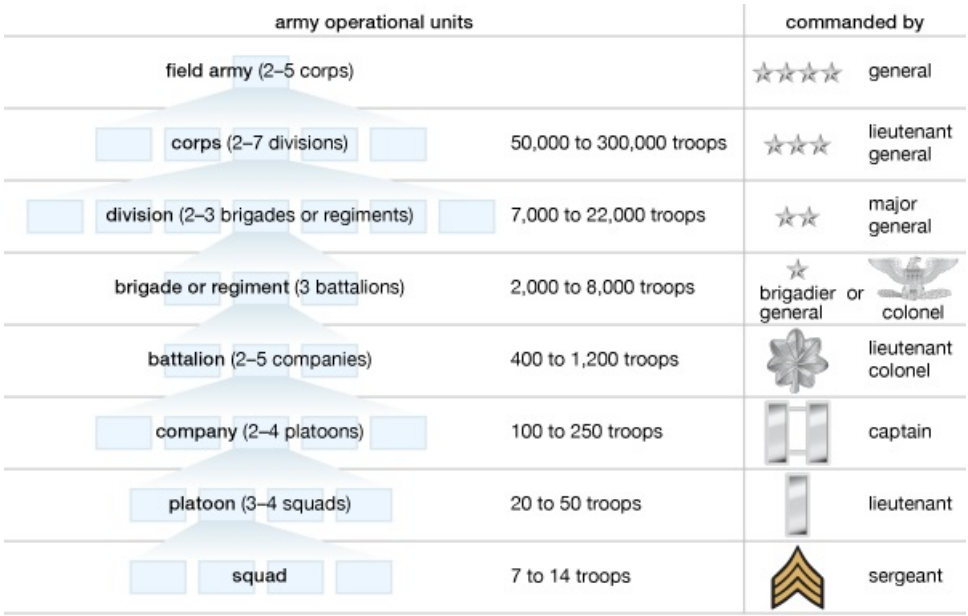
[31] E. Wayman, “What is war good for? ask a chimpanzee.”, 2012. [Online]. Available: http://www.slate.com/articles/health_and_science/human_evolution/2012/10/chimpanzee_wars_can_primate_aggression_teach_us_about_human_aggression.html.

[32] “What is a table of organization and equipment?”, [Online]. Available: https://www.globalsecurity.org/military/library/policy/army/toe/toenum.html.

[33] L. Zappa and R. Minneman, “Tactical vs strategic vs operational”, Board Game Geek, 2010, https://boardgamegeek vs-strategic-vs-operational, 08/12/2016.

# A

# Military Organization



| army operational units | | commanded by |
|---|---|---|
| field army (2–5 corps) | | ★★★★ general |
| corps (2–7 divisions) | 50,000 to 300,000 troops | ★★★ lieutenant general |
| division (2–3 brigades or regiments) | 7,000 to 22,000 troops | ★★ major general |
| brigade or regiment (3 battalions) | 2,000 to 8,000 troops | ★ brigadier general or colonel |
| battalion (2–5 companies) | 400 to 1,200 troops | lieutenant colonel |
| company (2–4 platoons) | 100 to 250 troops | captain |
| platoon (3–4 squads) | 20 to 50 troops | lieutenant |
| squad | 7 to 14 troops | sergeant |

**Figure A.1:** The various army operational units. It should be mentioned that depending on the country, the number of troops can change. Source: Encyclopædia Britannica

# B

# Simulators Comparison

The following table is the result of the study and comparison of different military simulators currently existing on the market.

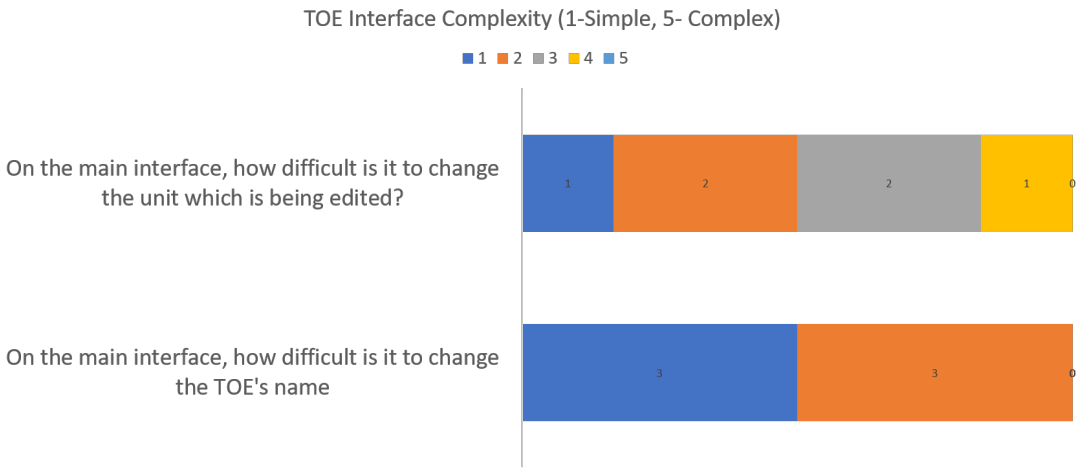
**Table B.1:** Comparison between military simulators

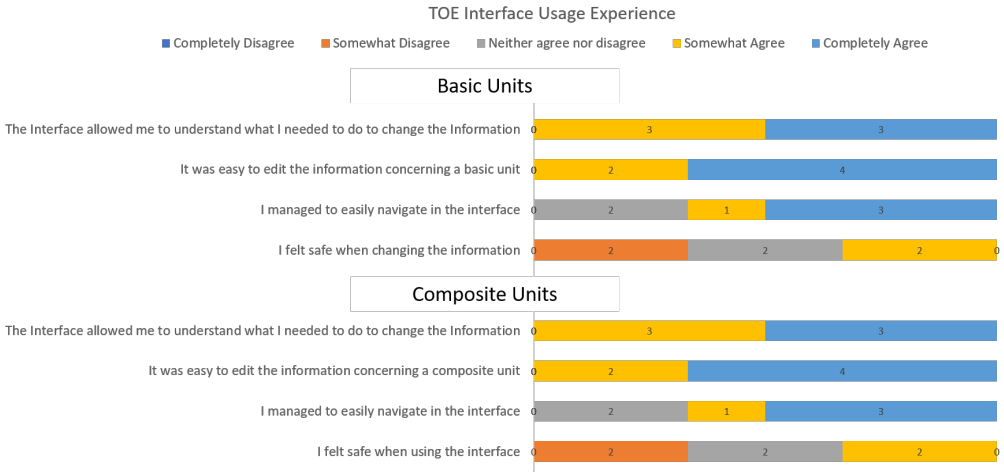
| **Name** | **Tac Ops 4** | **VBS Tactics** | **Sword** | **MÄK CST** |
|---|---|---|---|---|
| **Type** | Simulator | Simulator | Simulator | Simulator |
| **Company** | Battlefront.com | Bohemia Interactive Simulations | Masa Group | VT MÄK |
| **Command Scales Available** | Battalion or Regimental | Group to Company | Company or Battalion | Company to Division |
| **Possible matches** | Solitaire or multiplayer, with up to 8 teams | Solitaire or multiplayer, with as many teams as needed | Solitaire or multiplayer, with as many teams as needed | Solitaire or multiplayer, with as many teams as needed |
| **Maximum Connections** | 19, between spectators and players | Limited to the server latency | Limited to the server latency | Limited to the server latency |
| **Roles available** | Commander or Umpire | Commander or Instructor (Umpire) | Commander or Supervisor (Umpire) | Commander, instructor, executive officer, intelligence, operations, fire support, air defense, mobility or logistics |
| **Internal Communications** | None | Radio | Radio | Radio through HLA and DIS |
| **Multiplayer Architecture** | Clients join the umpire session | Dedicated server or by joining a session one client is running. | Dedicated server | Client join the instructor's session |
| **Scenario edition** | Yes, but fully manual | Yes, importing information from geospatial sources | Yes, importing information from geospatial sources | Yes, importing information from geospatial sources |

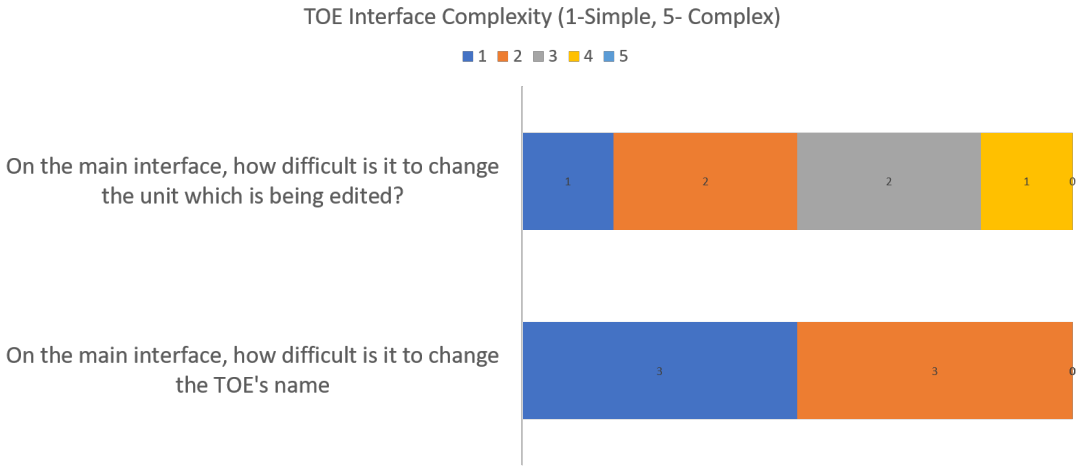
# C

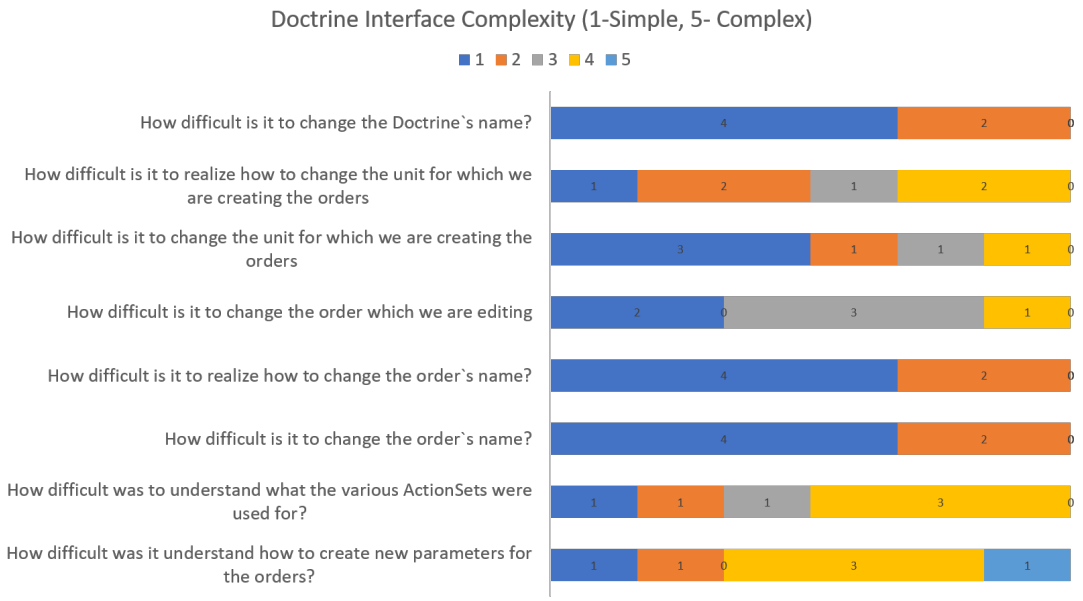
# Result of the questionnaires

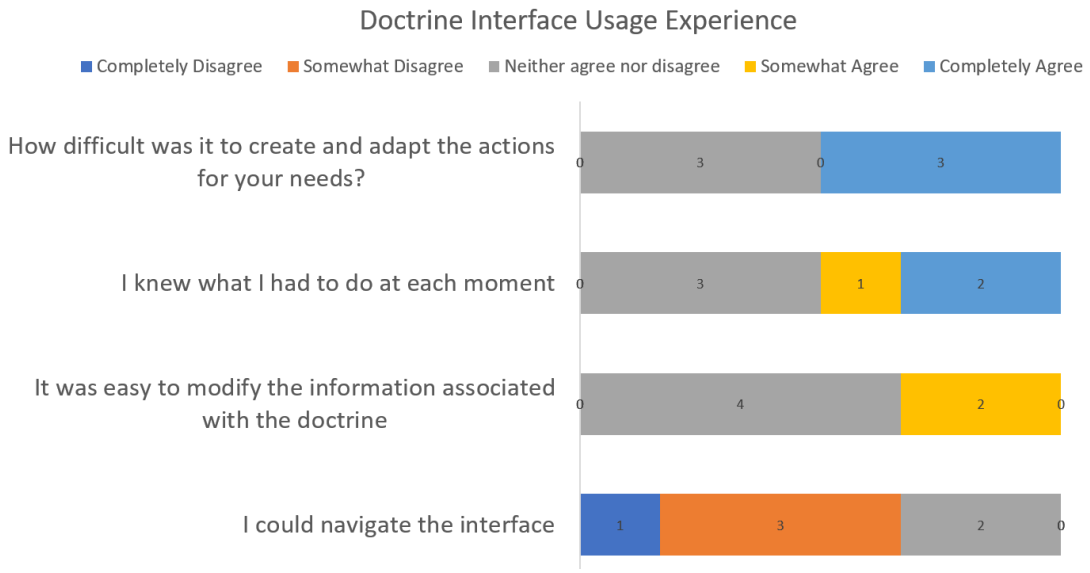
## C.1 Unit Information Questionnaire

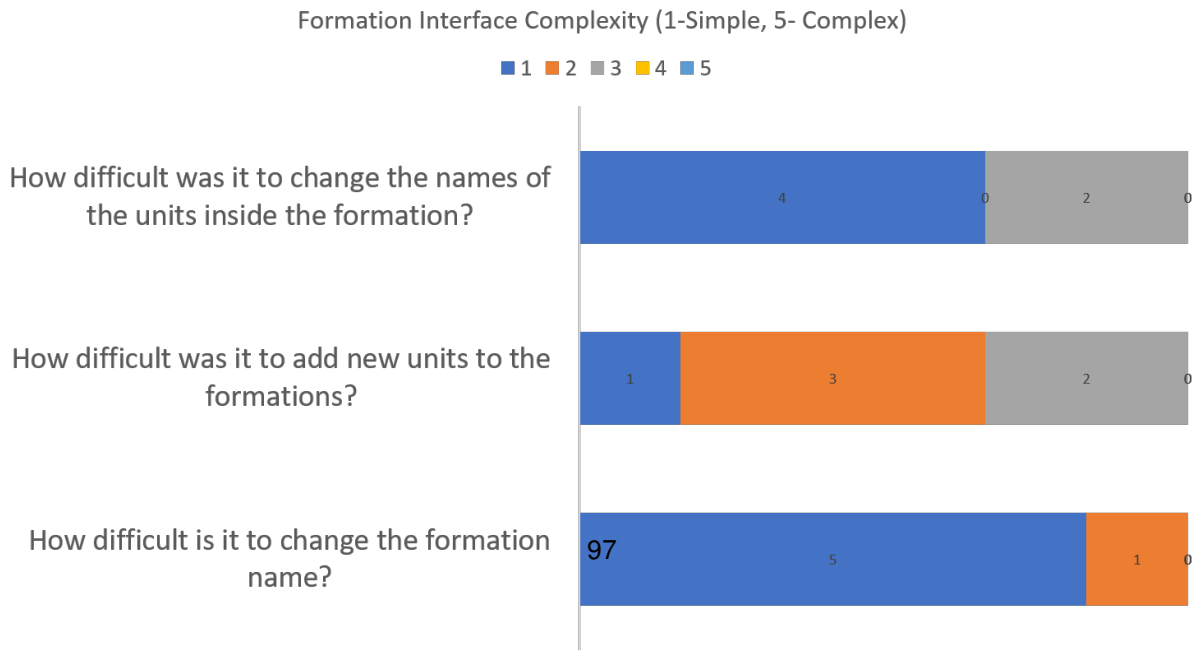


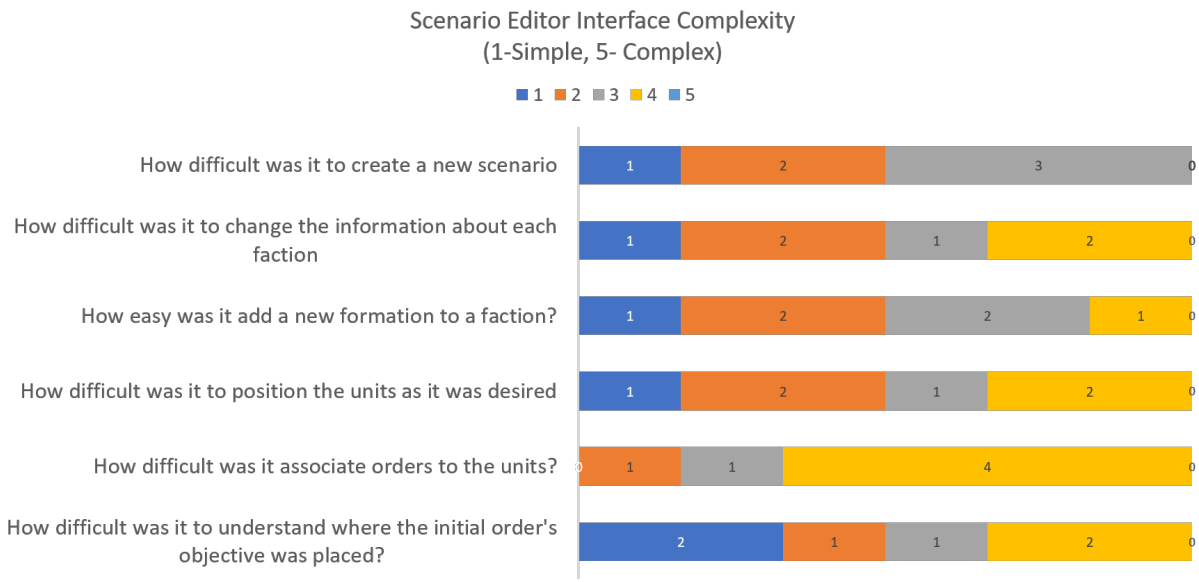



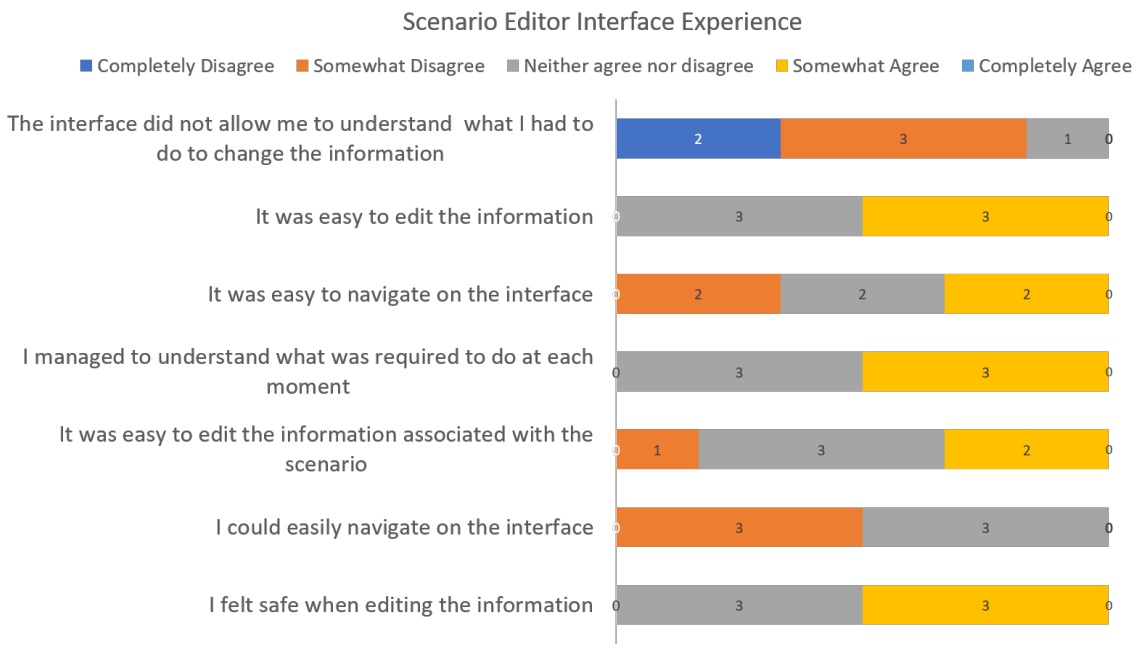

Doctrine Interface Complexity (1-Simple, 5- Complex)
1 2 3 4 5
How difficult is it to change the Doctrine`s name?
How difficult is it to realize how to change the unit for which we are creating the orders
How difficult is it to change the unit for which we are creating the orders
How difficult is it to change the order which we are editing
How difficult is it to realize how to change the order`s name?
How difficult is it to change the order`s name?
How difficult was to understand what the various ActionSets were used for?
How difficult was it understand how to create new parameters for the orders?


Doctrine Interface Usage Experience
Completely Disagree
Somewhat Disagree
Neither agree nor disagree
Somewhat Agree
Completely Agree
How difficult was it to create and adapt the actions for your needs?
I knew what I had to do at each moment
It was easy to modify the information associated with the doctrine
I could navigate the interface

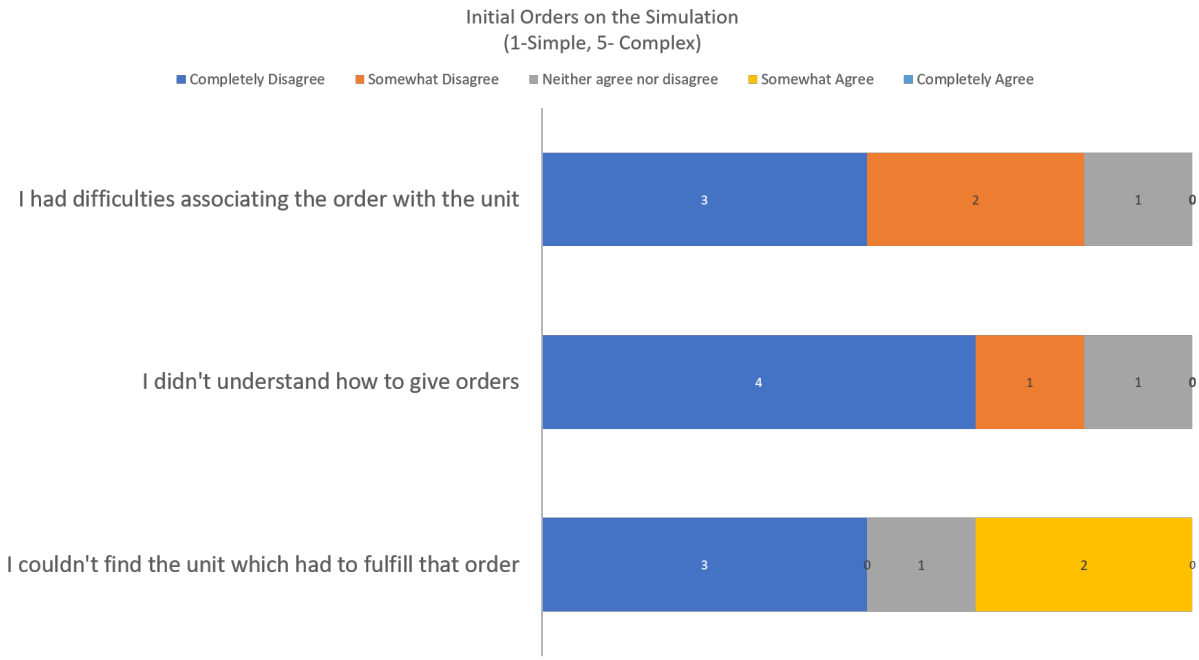

Formation Interface Complexity (1-Simple, 5- Complex)
1 2 3 4 5
How difficult was it to change the names of the units inside the formation?
How difficult was it to add new units to the formations?
How difficult is it to change the formation name?

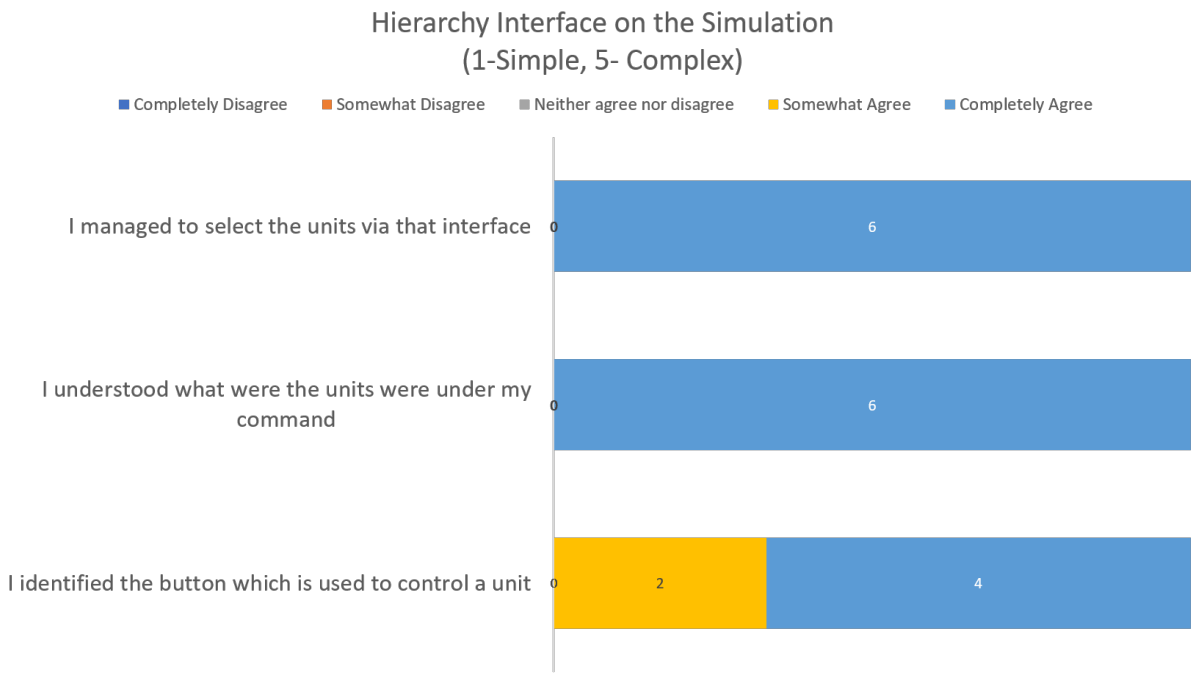
## C.2 Scenario Editor Questionnaire

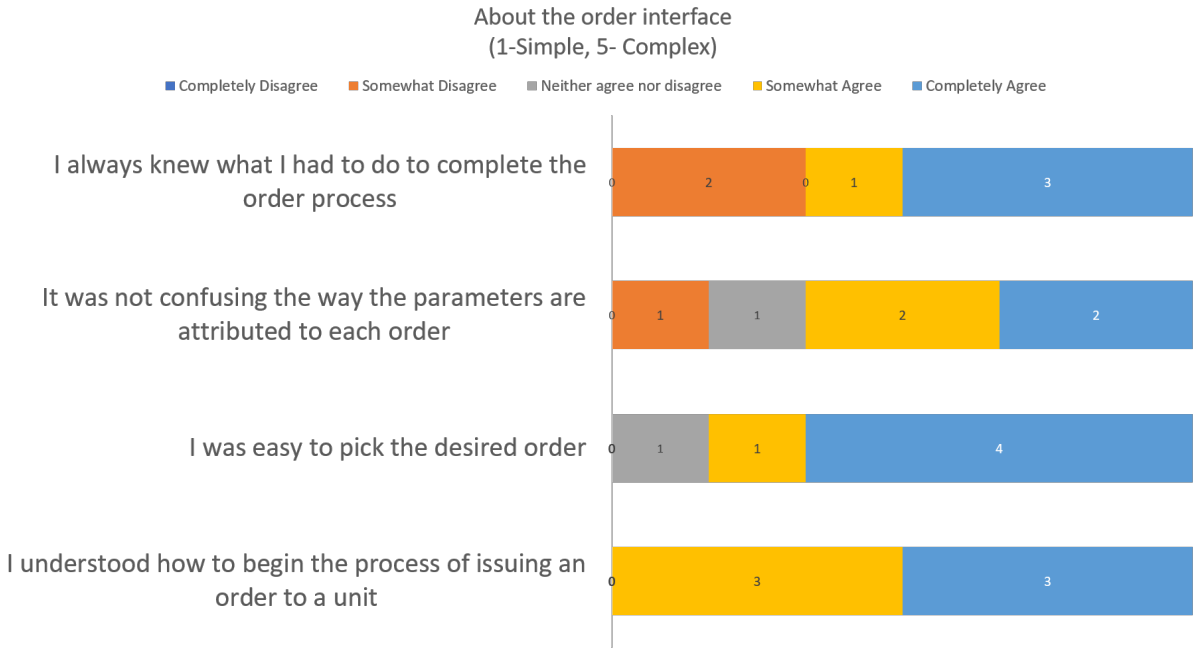


## C.3 Simulation Questionnaire




About the order interface
(1-Simple, 5- Complex)
Completely Disagree
Somewhat Disagree
Neither agree nor disagree
Somewhat Agree
Completely Agree
I always knew what I had to do to complete the order process
0
2
0
1
3
It was not confusing the way the parameters are attributed to each order
0
1
1
2
2
I was easy to pick the desired order
0
1
1
4
I understood how to begin the process of issuing an order to a unit
0
3
3