Ballistics Simulation Within Arma 3

Final Report

Submitted for the BSc in
Computer Science

May 2017

by

Sean Andrew Sidney Phillips

Word Count: 12,226

Contents

1 Introduction

Arma 3 is a military tactical shooter video game developed by Bohemia Interactive. The game takes place on the fictional Aegean islands of Altis and Stratis, with the former featuring 270 Km^2 of landmass and the latter 20 Km^2. Arma 3 consists of over 40 weapons, 20 vehicles and a multitude of other miscellaneous objects, such as clothes, buildings and ammunition crates – all contributing to generating an immersive and authentic experience. This, along with the realistic gameplay allowed by the use of the Real Virtuality 4 engine, creates a gaming environment closely following that of real modern combat.

In the modern age, tanks and other armoured vehicles play an essential role in the dynamics of combat and, due to Arma 3's aim of simulating modern combat, this follows suit in the gameplay of Arma 3. However, Arma 3's model of simulating armoured warfare is currently lacking in the accuracy of its simulation of projectile ballistics, adversely affecting the gameplay. This leaves a jarring experience when switching from more well fleshed out aspects of the game, such as the infantry combat. As Arma 3 takes pride in it authentic gameplay, "Experience true combat gameplay in a massive military sandbox. Authentic, diverse, open - Arma 3 sends you to war" (Bohemia Interactive, 2017), this only further exasperates the problem for the gaming community that regularly play Arma 3.

There is a considerable following of organised groups of players who get together and play Arma 3. This is how Arma 3 is reputed, by many, to be best played, "Join one of 1911 units and experience Arma 3 at its finest" (Bohemia Interactive, 2017). These groups usually play the game by mimicking real life combat tactics. Players are often split into teams with leaders and a hierarchy at play, all collaborating in an organised manner to defeat a common enemy, be that another player team or AI bots in user made missions. Many of these groups take the game very seriously - even calling themselves "MilSim" (military simulation) groups and naming themselves after regiments that exist in today's armies. They enjoy a high level of simulation and use mods such as "Advanced Combat Environment 3" and "Advanced Combat Radio Environment" to enhance the detail of simulation within Arma 3 to fit and meld the gameplay of Arma 3 to their own needs.

Arma 3 also has a substantial "modding" community that adds extra content, new features to compliment the realistic gameplay that Arma 3 provides and expand the existing features to a higher level of simulation. For example, the popular mod "Advanced Combat Environment 3", augments the infantry combat by revamping the medical system to a higher level of realism. However, within the "modding" community, there is a significant lack of focus on the ballistics simulation of projectiles hitting armoured targets when compared to other aspects of the game. This project will attempt to fill that gap and provide a package that brings the level of simulation at least on par with the rest of the game, all the while remaining within the original vision that the developers had for Arma 3.

The overall goal for this project is not too provide an exact physics simulation of projectile ballistics, but to improve upon the existing model, bringing up the quality of simulation to the rest of the core features that exist within Arma 3.

2 Aims and Objectives

To help manage the progress of this project, several objectives will be created to provide a criterion upon which the final completed package can be compared against. This will allow for an easy evaluation to determine if the final product has fulfilled the original goal.

The overall aim for the project is “To improve the level of simulation of projectile ballistics in Arma 3”. To achieve this goal, the project will have to complete the following objectives and meet their criteria.

1. **Retrieve from the game engine the properties of a projectile and its target after impact.**
2. **Interface with an external library and pass data between said external library and the game engine.**
3. **Parse the data outputted into an external library and store as variables for use within the library.**
4. **Calculate the penetrating ability of the projectile after impact and update the game engine appropriately.**
5. **Split armour values of the target vehicle based on the position of impact.**
6. **Create a module-based damage system where individual modules of the target vehicle can be damaged.**

2.1 Objective 1 – Retrieve from the game engine the properties of a projectile and its target after impact.

When a projectile impacts any vehicle in the game an event is required which returns the properties of the target vehicle and projectile, this will be needed for later calculations to determine the post-impact properties of the vehicle and the projectile. These properties will then have to be stored as variables for later use. These will include, among others, velocity of projectile, angle of incidence between projectile flight path and armour plating, type of projectile fired, type of target vehicle and position of impact.

2.2 Objective 2 - Interface with an external library and pass data between said external library and the game engine.

To be able to use an extension for the bulk of calculations needed, an interface is required between the game engine and an external library. Communication between the extension and the game engine needs to be fast and efficient, with the minimal amount of time between projectile contact and possible penetration, of which the later calculations need to occur in.

2.3 Objective 3 – Parse the data outputted into an external library and store as variable for use within the library.

Data passed into the external library needs to be parsed into a readable form. Setting up this will require string manipulation to store the data passed into variables. These variables will then have to be stored as the relevant type corresponding to the nature of the data passed. This will provide a base for the calculations of the impact of a projectile that will occur in the later stages of this project.

2.4 Objective 4 - Calculate the penetrating ability of the projectile after impact and update the game engine appropriately.

Using the data stored within the external library, a calculation will determine whether the projectile is able to penetrate the target vehicle's armour. This will be done via a set of mathematical equations simulating the real-world mechanics of armour penetration after which, the game engine will appropriately update with the projectile's post impact values and those of the vehicle. These values will include, among others, the damage caused to the vehicle, projectile velocity and direction of projectile (if deflection has occurred). Once an equation has been developed, time must be spent on validating the accuracy of said equation by comparing results to that of real-world examples. The same variables will be used between the real-world examples and the equation validation tests.

2.5 Objective 5 - Split armour thickness values of the target vehicle based on the position of impact.

To properly simulate a real-world engagement, the armour thickness values of the target vehicle cannot be the same throughout the vehicle. The armour thickness of a combat vehicles is not often homogenous, generally a vehicle will have its thickest armour at the front and the weakest at the rear, with the side armour being somewhere in the middle. The same assumption can be applied to the turret of the target vehicle, if present, as well. By virtue of Arma 3 being based in the near future with the vehicles used within the game being currently employed by major armed forces today, accurate armour thicknesses throughout the tank may be impossible to retrieve because these variables could be classified. Instead, estimations will be made using multiple sources (images, post-combat reports etc.).

2.6 Objective 6 - Create a module-based damage system where individual modules of the target vehicle can be damaged.

To improve the level of simulation of the damage model of armoured vehicles, it is insufficient to only have the proper penetrations of the outside armour. In addition to the exterior penetration calculations an interior damage model, where individual key components of the vehicle can be damaged, is required. This will allow a more realistic approach to the damage caused upon a target vehicle after impact. These damageable components will include, but not limited to, ammunition stowage, engine, crew members and fuel tanks. As stated in objective 5, estimations will be needed to determine the tank module components and their locations. Care will be taken to properly develop the module damage system to accurately emulate what happens in real-life scenarios when a combat vehicle is hit.

2.7 Further Objectives

After conducting a mid-term review on the overall progress of this project, further objectives have been laid out to account for the extra development time gained from being ahead of the initial time plan. These objectives will not add any diverse new features, instead focusing upon improving the already existing work by increasing the level of simulation to further match real-world vehicular combat as close as possible. Both objectives equations will be validated against real-world vehicular combat examples, following what is stated in objective 4.

2.7.1 Objective 7 – Further develop the penetration calculation to account for HEAT (High-Explosive-Anti-Tank) shells.

HEAT shells are a common round type in use within Arma 3. At the current level, they are very poorly simulated, even falling short of the current kinetic round simulation. To rectify this a new algorithm will be developed alongside the already existing penetration calculations for kinetic-energy type penetrators, within the external extension already developed. HEAT shells make use of the Munroe-effect and are independent of projectile velocity and will require extra research into this field to devise an equation to calculate the penetration of said shells.

2.7.2 Objective 8 – Further develop the penetration equation to incorporate factors important to developing an accurate simulation.

The mathematical equations that have been developed in earlier objectives are more accurate than the base game of Arma 3, yet are still lacking in numerous aspects to properly simulate a real-world vehicular engagement. The equations in use by objective 4 omit numerous factors that happen in real-world ballistics. Some of these factors that could be considered are, the shape of the projectile head, the slope-effect (how a shells penetration is affected by amour sloping) and the perforation of armour that occurs during impact.

3 Background

In this project, there are many topics to research and understand involving the mechanics of projectile ballistics in real life and within the game before work on the project can commence. The problems underlying the base game of Arma 3, notably containing the tank warfare aspect of the game, will be highlighted and explained fully so that an appropriate solution can be developed - one that will meet with the project aims and objectives. Steps must then be taken to consider the technologies that can be used. Several probable algorithms will then be proposed, their key features critically analysed for the strengths and weaknesses, all in relation to the project as a whole.

3.1 Problem Context

Arma 3 simulation of modern warfare is arguably the most authentic gameplay experience on the market, however, there is a significant gap between the level of simulation between projectile ballistics and the other core features that exist within the game. Arma 3 fully simulates a projectiles flight path through the interior only for those that penetrate through the entire vehicle, any projectile that stops within the vehicle body will be treated as if it has stopped at its surface. A projectile's flight time within a target is not currently represented within the game engine. There are also no detailed physics models on projectile ballistics; different projectile types are poorly implemented and in some cases, they do not properly mirror the real world mechanics of projectile ballistics.

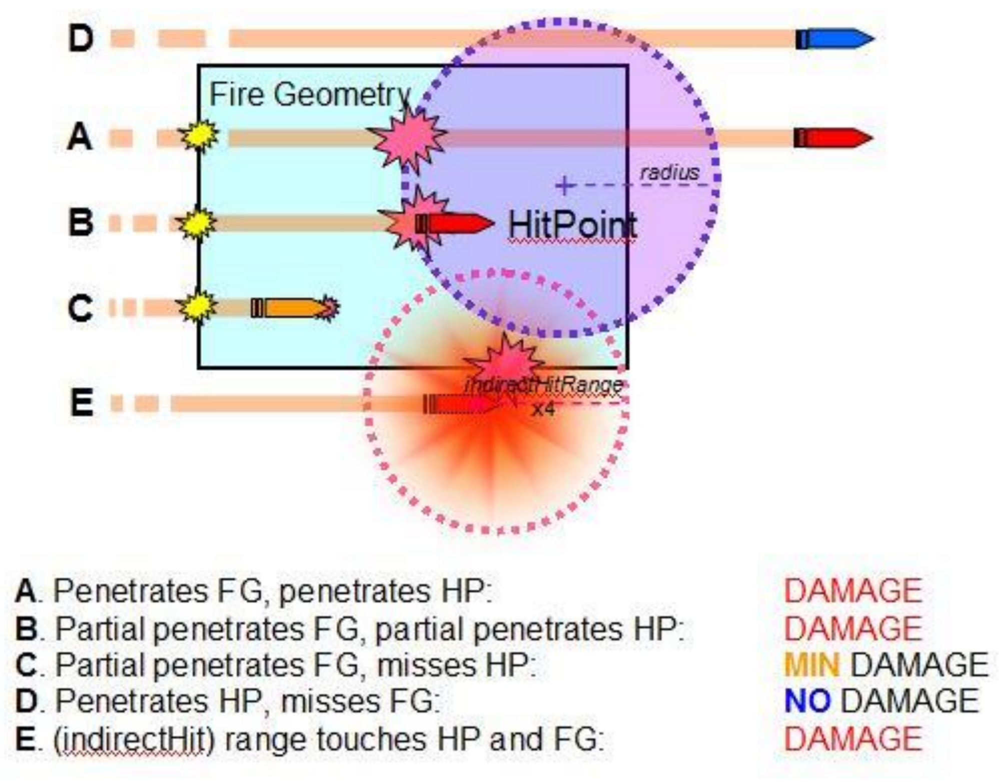
Overall, Arma 3 does partially represent a projectiles penetration of an armoured target, but its significance is overwhelmed with the usage of a global health pool that allows damage to a vehicle with no penetration occurring.

3.1.1 Current Model of Projectile Ballistics Within Arma 3

Arma 3 uses a basic hit point based damage system to model the vehicle warfare, involving vehicles and infantry firing projectiles at armoured target vehicles. The game uses two systems to determine the damage done to a vehicle, these are fire geometry and hit points. Fire geometry is an invisible physical model acting as the armour plating for the vehicle – this is used to calculate if a projectile has intersected with the vehicle model. Hit points are used to establish the global and individual modules health pool for a vehicle, also determining the location of damage, damage done and to what component of the vehicle.

Once an impact has occurred, the penetration and damage are not one and the same within the model that exists. An attack must simultaneously penetrate fire geometry and a hit point radius or, an explosive radius from a projectile intersects with a hit point spheres radius. This is where problems can occur, as if the hit point spheres are laid out to where they extend outwards past the fire geometry, damage can be dealt without a projectile penetrating or even impacting a section of fire geometry. This will mainly occur with the use of explosive rounds as the explosive radius can intersect with a hit point sphere causing damage.

When the game engine comes to calculating the penetration of a shell against a piece of armour, it is very simplified – only using the shell velocity, calibre and the armour thickness. There is no usage of other material properties (such as density of the projectile / armour, shape of the projectile etc.) needed to fully represent the real-world mechanics of projectile ballistics.



A. Penetrates FG, penetrates HP:	DAMAGE
B. Partial penetrates FG, partial penetrates HP:	DAMAGE
C. Partial penetrates FG, misses HP:	MIN DAMAGE
D. Penetrates HP, misses FG:	**NO** DAMAGE
E. (indirectHit) range touches HP and FG:	DAMAGE

Figure 1. A diagram of a small magnified cut out of a vehicle armour and interior further explaining damage and penetration to fire geometry and hit points and how those damage the vehicle – (FG = Fire Geometry / HP = Hit Point), (Olds, 2015)

In figure 1 attacks A-D are direct hits and attack E is an example of an indirect hit. This is where problems can occur and the damage system does not simulate real life closely – a shell with a large indirect hit range can bypass the fire geometry and damage the hit point radius. The diagram above also shows that the fire geometry acts as a sort of armour, protecting the vehicle from direct attacks, however (as stated above) indirect fire can bypass the fire geometry and still cause significant damage to the vehicle.

Typical Damage Process

To further explain how Arma 3 damage system works, this is a step-by-step process of how damage usually transpires in Arma 3.

1. Initial Impact

Firstly, a projectile has impacted the armour of a vehicle but has not penetrated. Regardless of whether the round has penetrated the armour, the vehicle takes some initial damage. Locations nearest the impact will take the most damage with the amount falling off geometrically. The vehicle global health pool will also take damage, in accordance to the physical properties of the projectile, even allowing the vehicle to be destroyed if the damage taken is significant enough and all without the projectile penetrating the vehicles armour.

2. Penetrating Damage

Once the projectile has been determined to penetrate the armour of the vehicles, it will continue through the insides of the tank, damaging the vehicle modules as it goes. As the projectile travels through the tank it will significantly generate more damage to the surrounding modules as well as still damaging the global health pool.

3.1.2 Mechanics of Real-World Projectile Ballistics

To properly develop a solution for this project, one must first understand the real word mechanics of projectile ballistics and what can happen to a target vehicle that has been impacted by a projectile. The types of projectile that can threaten an armoured vehicle are usually separated into two main groups, specifically kinetic and chemical energy weapons.

Kinetic Energy Weapons

The most common category of kinetic energy weapons that exist within Arma 3 are long-rod penetrators, namely an APFSDS (Armour-Piercing-Fin-Stabilised-Discarding-Sabot) shell. This type of shell is mainly fired from a gun mounted on a main battle tank and rely upon kinetic energy alone to destroy an armoured target. An APFSDS shell is a long, thin rod of a dense material, usually a tungsten-carbide or depleted uranium alloy, with a larger, thicker and soft jacket that surrounds the rod core – allowing the shell to be fired from a barrel of a much larger diameter than the core. After leaving the barrel of a gun the dense core will eject the surrounding jacket and continue on it flight path towards the target.

Upon impact with an armoured vehicle, the shell will rely upon it kinetic energy to force its way through the armour. If it is successful at defeating the armour then the shell will generate heat and spalling, creating a pressure wave that will travel through the vehicle, destroying the target. If the projectile does not have sufficient kinetic energy to defeat the armour then the shell will either shatter upon impact or deflect away harmlessly.

Chemical Energy Weapons

Unlike kinetic energy weapons explained above, chemical energy weapons rely upon the release of energy from an explosion and the resulting affects to inflict damage on armour. These are represented within Arma 3 by the use of HEAT (High-Explosive-Anti-Tank) and HE (High Explosive) shells. Both of which are not only fired from armoured vehicles, but employed also by infantry, helicopters and airplanes.

HEAT shells operate on the fundamental basis that the Munroe Effect provides. The Munroe Effect is when a conical or hemispherical hollow is shaped into the front end of an explosive charge, greatly increasing the penetrating ability of any explosive charge. HEAT shells have a cavity and a metallic liner facing towards the target and, upon impact, the explosive charge within the shell will detonate, collapsing the metallic liner forming a high-velocity jet of molten metal (typically copper) that will melt its way through the armour.

HE rounds rely solely upon the blast wave formed from the impact explosion and the subsequent flying fragments of the shell. Detonation will occur upon impact with a hard surface, due to an impact fuse located at the front end of the shell's body. The blast wave and fragments formed upon impact are often ineffective against well-armoured targets and will often fail to penetrate a vehicle armour. However, the shell will still generate a pressure wave within the target vehicle, even if no penetration has occurred, inflicting x damage upon the crew and essential components.

3.2 Comparison of Technologies

To create a simulation of projectile ballistics and their interactions with armoured vehicles there are a number of technologies that can be used to achieve this. This is in reference to the game engine that can be used, as there are alternatives with sufficient resources in addition to Arma 3. These technologies with their advantages and disadvantages from their implementation will be assessed as to confirm the right solution for this project.

To continue development of this project there are numerous technologies which can be used. These include modifying an already existing software package (such as Arma 3 and VBS 3) or starting from the ground up and using a basic third party game engine such as Unity or Unreal

3.2.1 Arma 3

The technology that is the most developed in the area of this project would be the Arma 3 engine. This encompasses a package that is already fully developed and incorporates a simplified simulation of projectile ballistics and vehicle combat along with the other features of the game that compliments this. However, having a partially developed solution can pose a number of problems along with the obvious benefits that it provides.

The advantages to using the Arma 3 game engine would include:

- A vehicle combat system is already in place within Arma 3. This implicates less development time spent on modelling in the basics of vehicle combat, such as driving and shooting, which would be out of scope of the project aims and objectives.
- There are various coding languages that can be used for development within Arma 3, offering a number of alternative methods if the software engineer is not familiar with a particular language.
- Arma 3 offers a number of testing functions to assist in black box testing. Vital development time can be spent in other, more important areas that are relevant to the projects objectives.
- Arma 3 is easily accessible being a popular PC video game sold from a multitude of sources.

On the other hand, developing with Arma 3 does not come with its faults. The disadvantages that may occur are:

- Arma 3 lacks any significant documentation and a multitude of experimental investigations into the game itself, to familiarise with the software's functions, will need to occur before any real development can begin.
- There are poor error checking systems in place when conducting white box testing; No stepping through code exists and script errors are often obscure and inaccurate.
- If the current projectile ballistics simulation is not in line with the projects goals, then development time will be wasted on workarounds to bypass the code that already exists.

3.2.2 Third Party Game Engine

A game engine developed by a third party can be considered for use with this project. There are many options to choose from, however the most popular and arguably considered the best to use would be the Unity and Unreal game engines. Using these engines would offer a number of benefits, most notably the ease of development. Multiple error checking systems are in place helping speeding up the development process.

Advantages from developing with a third-party game engine include:

- These third-party game engines are easily accessible, often being free if development is done without the intent of monetising the final product.
- They are very well documented with numerous tutorials and articles on the systematics of the software – less development time would be spent on learning the new software tools.
- When conducting white box testing there are numerous error checking tools available, making the process quick and efficient.

There are many disadvantages from developing with a third-party game engine however:

- A basic vehicle combat system would have to be developed before the work can begin on the projects objectives, meaning vital development time wasted.
- Testing functions would have to be created to assist in black box testing.

3.2.3 VBS 3

VBS 3(Virtual Battlespace 3) is a virtual military battle simulator developed by Bohemia Interactive Simulations that is related to Arma 3 by being derived, at a basic level, from the same engine. Due to this, many of the advantages and disadvantages are the same as Arma 3 with only a few variances.

Advantages from using VBS 3 would be:

- Even more development would be done in relation to the project's goals, implying less time spent on bringing the current model up to par before real project work can commence.
- A multitude of languages are natively supported, not unlike Arma 3, allowing various alternative solutions to the project's objectives.
- A number of functions exists that can be of assistance when conduction black box testing.

Disadvantages would include:

- A significant lack of documentations, even when compared to Arma 3, due to VBS 3 being a closed piece of software only available to large institutions.
- Development tools are limited with few error checking systems in place, white box testing would be time consuming as a result.
- Limited access to software as it is only available to certain customers such as universities and those with military related backgrounds.

3.2.4 Decision

To decide the final technology to be used for development of this project a multitude of factors will be taken into consideration, such as performance and efficiency, ease of development and extent of documentation available. The engineers experience will also have to be taken into account along with those outlined in the above.

After considering the factors evaluated, the technology to be used with the development of this project will be Arma 3. The software engineer is familiar with this piece of software and has vital experience in development related to Arma 3. Arma 3 does come with its disadvantages however, such as a lack of documentation and error checking systems, but these are out-weighed from the gain in development time that would be saved from not having to create a basic vehicle combat model – out of scope with the projects aims and objectives.

3.3 Alternative Solutions

After confirming the technology to be used, there are a number of methods that can be used to achieve the projects aims and objectives. These are in relation to the coding language that will be used and whether to use the inbuilt scripting engine or use an external library and establish an interface between this and the game engine.

3.3.1 SQF

The in-built scripting language is an obvious choice for development at first glance. It comes pre-built within the engine and has a multitude of commands available for use within the game. Development with SQF however, is made harder due to the lack of documentation available. It is a relatively simple language syntactical, but does not offer good performance and efficiency. Using SQF incurs a lack of error checking systems in place to assist with white box testing and debugging.

3.3.2 C++ Extension

A C++ extension will require further development to set up an interface between the game engine and the extension but it offers a number of benefits. These include increased performance and a more efficient development environment with the usage of Visual Studio. However, the application of a C++ extension to this project does come with a downfall. This is because it is a lower-level language syntactically and can be un-wieldy to use.

3.3.3 C# Extension

Not unlike the C++ extension, a C# extension will require further development to work with the game engine and to be able to communicate effectively. Like the C++ extension there are multitude of benefits to be had with using a C# extension. These include increased performance when compared to SQF, however not as fast as compared to using a C++ extension. It has a more efficient development environment (again with the use of Visual Studio) and, being an intrinsically higher level language than C++, is an easier language to develop with.

3.3.4 Decision

To make the decision for the technology to use for the development of this project, a multitude of factors will need to be taken into consideration. These will include the software engineer's familiarity with the language, the syntactical complexity of the language, difficulty and time to set up development and the efficiency and features of the development environment.

After consideration of these factors the technology to be used is a C++ extension. This is due to the considerable performance benefits and features of a proper integrated development environment - being able to develop completely outside of the game without lengthy boot-up times to test code will be invaluable for the shortening the development process for this project. Likewise, features such as breakpoints will shorten the time spent debugging the code. There are downfalls to developing using C++ when compared to SQF and C# however, as it is a more complex language syntactically and incurs a possible lengthy set up process. Regardless, these are easily overcome due to the engineer's familiarity with C++.

3.3.5 Third-part developed Modifications

Arma 3 has a significant modding community which has created modifications that can streamline a modifications development process. One of such modifications is Community Based Add-ons (CBA). This modification provides a number of development tools that can be used to access functions, both originally within Arma 3 and newly developed by CBA, which would be otherwise impossible without significant workarounds, decreasing a projects performance and increasing development time. The section of CBA that will be used within this project will be extended event handlers. This section of CBA will allow a function to be called every time an event handler is fired, as opposed to having to manually tag said event handler along with every in-game unit that will be affected by the projects code.

3.4 Comparison of Algorithms

The most notable algorithms to be used within this project will be the penetration probability at impact and the post-impact values of a projectile and its target. There has been significant research in the academic world within this field, all of which, are vastly different in the complexity and accuracy of the equations derived from this research.

The algorithms that have been chosen are equations derived from academic research. These include the DeMarre equation, a formula for impact depth conceived by Sir Issac Newton and Project Thor, a technical report into projectile ballistics done by U.S Army contractors.

These equations have been chosen from the requirements of objective 1-6, objective 7 and 8 will have another section to determine suitable equations that fulfil the requirement of objective 7 and 8.

3.4.1 DeMarre Equation

The DeMarre equation was used by a multitude of countries militaries during and, for a time, after World War 2. It estimates the penetration of projectiles against homogenous steel armour and is assumed that the impact angle is 0° and the projectile does not deform. The DeMarre equation only applies to kinetic energy projectiles and those with no explosive nature. The DeMarre formula is as follows:

$$t = d^n \sqrt{\frac{wv^2}{kd^3}}$$

Where w = weight of projectile, lbf

V = velocity of projectile, ft/s

d = diameter of projectile, in

t = thickness of armour penetrated, in

k = constant, conforming to the projectile and target materials properties

n = 1.4

The use of this equation would provide a relatively accurate simulation of projectile ballistics and yet remaining simple enough to be within the project scope, however it does come with a number of shortcomings. Firstly, there is no consideration for the angle of impact and crude calculations would be needed to apply this affect to the equation. Second, there are no values given for the residual velocity after impact. Finally, the shape of the projectile is not taken into account. However, the formula would be simple to transcribe to code, with minimal mathematical functions needed.

3.4.2 Impact Depth

Impact depth is a rough approximation for the depth of penetration by a projectile originally theorised by Sir Issac Newton. As this equation is only a rough approximation for penetration it does have numerous drawbacks and would be generally unsuitable for anything but the simplest of calculations. Only a limited set of properties are considered - material density of the target, area of the impact and length of the projectile. The equation does not consider vital properties important to the performance of a projectile such as the angle of impact, shape and velocity of the projectile. Most notably, this equation only holds true for blunt projectiles and target materials with no cohesion.

$$P \approx L \frac{Dp}{Dt}$$

Equation to estimate a projectiles penetration depth.

Where P = Penetration Depth, cm

L = Length of Projectile, cm

Dp = Density of Projectile, g/cm^3

Dt = Density of Target material, g/cm^3

3.4.3 Project Thor

Project Thor is a technical report conducted by U.S Army contractors in 1961. The report gathered and analysed perforation data for steel fragments impacting on metallic targets. The fragments used were small (less than 10cm in diameter) and had a length/diameter ratio of less than 3. Three equations were derived from the research, these were the striking velocity to just penetrate, the residual velocity and residual mass after penetration. The equations described are as follows.

$$Vr = Vs - 10^{c} \, (hA)^{\alpha} \, (Ms)^{\beta} \, (\sec\theta)^{\gamma} \, Vs^{\lambda}$$

$$Vo = 10^{c1} \, (hA)^{\alpha 1} \, (Ms)^{\beta 1} \, (\sec\theta)^{\gamma 1} \, Vs^{\lambda 1}$$

$$Ms - Mr = 10^{c} \, (hA)^{\alpha} \, (Ms)^{\beta} \, (\sec\theta)^{\gamma} \, Vs^{\lambda}$$

Where Vr = projectile residual velocity after penetration, in ft/s

Vs = projectile striking velocity, in ft/s

Vo = projectile striking velocity, just to penetration, in ft/s

h = target material thickness, in inches

A = impact area, in inches

Ms = weight of projectile, in grains

Mr = residual weight of projectile after penetration, in grains

Θ = angle of incidence between projectile flight path and target material normal, in degrees

α, β, γ, λ = constants depending on projectile and target material

Project Thor encompasses three highly detailed equations for determining the projectile and target post-impact properties. It considers the angle of impact, coefficients unique to specific materials and even giving the velocity and mass after penetration has occurred. However, among these advantages there are a number of disadvantages that Project Thor possess. First, only projectiles with a length / diameter ratio less than 3 were used. Second, only projectiles were analysed if they did not deform or break up on contact. Third, notably the greatest disadvantage, is that small bullet-sized fragments were used. As this project development consider projectiles much larger in nature, it may produce inaccurate results when Project Thor's equations are applied to larger projectiles.

3.4.4 Decision

To determine the final algorithm in use for calculating the penetration probability and the post-impact values of the projectile and target a number of factors will be evaluated. The algorithm chosen must not be too complex, as this projects goal is not to reproduce an exact copy of real world ballistics. It must be easily transcribed to a coding environment and remain relevant with the projects aims and objectives.

The algorithm to be used for this project will be the DeMarre equation. This equation provides a more accurate simulation when compared to impact depth, it considers the velocity of a projectile and is based on projectiles with an ogival head, on the contrary, impact depth only considers projectile with a blunt nose. It is also derived from research done in the same area as this project, unlike that of Project Thor which is conducted on much smaller projectiles, further confirming that results gained from it will be accurate. The DeMarre equation maintains these advantages while also remaining simple and within the projects aim to bring the level of simulation up to par. It is not without its disadvantages however, as there are no considerations for the angle on impact, therefore further calculations will be needed to compensate for this deficit.

3.5 Algorithms for Advanced Penetration Equations

From what is stated in objective 7 and 8, new equations will be researched to complete said objectives. This involves the finding of a more advanced kinetic-energy penetrator equation that properly suits the demands of objective 7 and a formula to simulate a HEAT shell penetrator for objective 8.

3.5.1 Algorithm for Kinetic Energy Type Penetrators

The algorithm in use for early development done of this project will be the DeMarre equation, this equation does a good job of creating a simulation loosely based on real-world penetration mechanics. However, this equation is severely lacking in numerous aspects when analysed for the range of factors that are considered by this equation. The equation omits a number of important variables that can have considerable effect on the outcome of a scenario of a kinetic energy shell impacting against a vehicle.

The algorithm that has been chosen has been empirically derived from multiple firing tests conducted by W. Lanz and W. Odermatt for the 13th International Symposium on Ballistics. This equation accounts for a large range of variables that exist within the mechanics of real-world ballistics. Most importantly, the equation is directly derived from firing tests using the same tank shells that are present in Arma 3 – providing confidence that the results obtained from using this equation will be authentic and will closely follow a comparable real-world example.

$$\frac{d}{D} = \alpha\left(\frac{L}{D}\right) \times cos^{0.745}\theta \times \sqrt{\frac{P_p}{P_T}} \times e^{-\frac{25.9 \times R_m}{P_p \times v_T^2}}$$

Where

L = Projectile Length, in mm

D = Projectile Diameter, in mm

P_p = Projectile density, in kg/m^3

V_T = Projectile impact velocity, in m/s

d = Target Plate thickness, in mm

R_m = Tensile Strength of target plate, in N/mm^2

Ɵ = Angle of incidence between shell and target plate, in degrees

P_t = Density of target plate, in kg/m^3

3.5.2 Algorithm for Chemical Energy Type Penetrators

A new algorithm will have to be developed to meet the requirements of objective 8. This includes accurately simulating HEAT shell penetration of a target vehicle. To achieve this an equation will be chosen that provides an accurate representation of a real-world HEAT shell impact yet remaining within the project scope.

The equation that will be used has been developed by Ogorkiewicz and uses 3 main variables to deduce the penetration depth of a HEAT shell. These variables are the length of the molten metal jet within the HEAT shell, the density of said metal and finally, the density of the target material. Using this equation to continue development on this project will provide a number of benefits, most notably it being an accurate equation yet remaining relatively straightforward to implement.

$$t = L\sqrt{\frac{P_j}{P_a}}$$

Where

t = depth of penetration, in mm

L = length of molten metal jet, in mm

P_j = density of molten metal jet, in kg/m^3

P_a = density of target material, in kg/m^3

3.6 Processes and Methodology

To commit to development of this project the most suitable process that can used would be the prototyping method. The prototype process is where first a suitable prototype is developed, done by gathering the requirements for the prototype, and then designing the code to be developed before actual coding of the prototype can begin. Once the prototype has been developed to a testable state, testing will occur on said prototype. The feedback gained from the testing will be analysed and, if needed, work will be done on refining the current prototype to meet the prototypes criteria, before finally moving on and continuing further development of the next prototype.

This applies well to this projects development cycle as each objective can be completed and tested in its own environment without relying upon latter objectives to work. This would work in practice by making each objective stated in section 2 as a prototype in off itself. To develop a prototype, research will be done on how to meet the objective, in turn plans will be created on how to implement the relevant objectives criteria into coding, before finally carrying out the coding process. After a suitable prototype has been developed, testing will occur to determine whether the most recently developed prototype matches the criteria of its corresponding objective and, if the objective has been completed, work can continue on developing a prototype for the next objective. If testing does show that the objective has not been suitably met, then the cycle of refining the prototype and testing will repeat until the criteria of the relevant objective has been met.

This section will now break down each prototype that will be developed in the working of this project. Simple, basic tests will also be mentioned that will conclude if the prototype is in a workable state (not to be confused with testing that will be designed in section 4 to test the final products validity).

3.6.1 Prototype 1 – Retrieve from the game engine the properties of a projectile and its target after impact

This prototype will be developed using objective 1s (in section 2.1) criteria as a base for which to make plans for development. To achieve this, an extended event handler, via the use of the Community Based Addons modification, will be used. This event handler, namely the “hitPart” event handler, will fire upon an impact between any projectile and a vehicle. Upon firing, a function will be executed. The properties of the projectile and vehicle will be interjected into variables allowing for use within latter calculations. To conduct testing upon the initially developed prototype, the variables retrieved will be outputted into an external text file granting a comparison of said variables to expected ones.

3.6.2 Prototype 2 – Interface with an external library and pass data between this and the Arma 3 game engine

Development of this prototype will be based upon objective 2s (in section 2.2) criteria. To conduct development of the interface, a C++ extension will be used and a multi-threaded solution, using third-party developed code by “KillZoneKid”, will pass data between the external library and the game engine on another thread. This will offer the advantage for the game engine to resume its activities until the external library has finished executing its code and is ready to send the resulting data back along the interface. Testing will be done by passing the properties, retrieved in prototype 1, along both ways of the interface. The data will then be analysed within the game engine and the external library to verify the validity of the data passed and that it hasn’t been modified in any way.

3.6.3 Prototype 3 – Parse the data outputted into the external library and store as a variable for use within the library

This prototype will use objective 3s (in section 2.3) conditions as a basis for development. To continue with the expansion of external library’s features, the data that was retrieved in prototype 1 will be parsed into the appropriate types, all done by the use of standard C++ parsing practices. To validate the variables parsed, these variables will be outputted into a separate text file and compared to that of the properties retrieved in prototype 1.

3.6.4 Prototype 4 – Calculate the penetrating ability of the projectile post-impact and update the game engine appropriately

The criteria laid out in objective 4 (in section 2.4) will be used as a foundation for this prototypes development. This prototypes development will focus on implementing the algorithms decided for use within section 3.4, namely the DeMarre equation. This equation will be coded into the C++ extension and will use the properties parsed into separate variables within prototype 4. Upon completion of this, the updated properties of the vehicle and projectile, with the fitting damages applied, will be sent back into the game engine along the previously developed interface. Parsing facilities will be developed within the Arma 3 game engine as only a single string data type can be sent along the interface. A test environment will then be built with specific circumstances which, when run, the penetration of the projectile, it’s properties and those of the vehicle, can be compared to how the situation has been calculated on paper.

3.6.5 Prototype 5 – Split armour thickness values of the target vehicle based upon the position of impact

The conditions specified within objective 5 (in section 2.5) will be used to make plans for development of this prototype. To allow for armour thickness values to be changed at varying locations on the vehicles surface, a database of sorts will be set up with the vertices locations of the target vehicles fire geometry. Development has to be conducted this way as Arma 3 does not include any methods for retrieving the armour thickness at an impact point. The armour thickness values can then be retrieved upon impact between a projectile and its target vehicle by comparing the position of impact to the armour thickness location database. Once retrieved, these values can then be plugged into the algorithm developed within prototype 4. Testing will be done by setting up test environment, with a projectile impacted an armoured vehicle. After which, the location of the impact will be compared to a database containing the armour thickness value at each location of the tank. The location will be noted and the armour thickness value retrieved can be compared to what is expected at the particular point of impact.

3.6.6 Prototype 6 – Create a module based damage system where individual modules of the target vehicle can be damaged

This prototype will use the criteria set out by objective 6 (in section 2.6) as the groundwork for its development. Creating a module-based damage system is where the project becomes significantly more complicated, as the projectile flight path and any subsequent impacts that occur within the vehicle are all developed within the C++ extension. The module system will be developed in a comparable state to that of prototype 5, with the location of the projectile being compared to a database of each modules vertices locations. The projectile flight path and deflections within the tank, if they occur, will all be simulated within the C++ extension and only the final output when the projectile stops, or the vehicle is destroyed, will be sent back to the game engine to implement the changes to the projectile and vehicle properties. Testing on this prototype has also significantly increased in complexity when compared to earlier prototypes. However, to carry out testing all the projectile locations and damages at set intervals will be recorded and logged into a set of graphs, allowing clear visual feedback of what has happened in the test.

3.6.7 Prototype 7 – Advanced penetration equations of kinetic and chemical energy type penetrators

The requirements put forth by objective 7 and 8 (in section 2.7 and 2.8 respectively) will be used as a foundation for this prototypes development. To conduct the development of this prototype, the plans laid out within prototype 4 will also be used as a basis – due to how similar they both are to each other, with the only difference being the complexity of the equations used and, in the case of objective 8, the projectile type being used. In light of this, testing will be similar to that of prototype 4 – a test environment will be set up, the post impact projectile properties and those of its target vehicle will be outputted into a text file and finally, will be compared to expected values calculated beforehand on paper.

4 Technical Development

This section will examine the development process and the implementation of the projects main goals. First, a look at the initial development work done to set-up the project base, then a look at the experiments conducted to gain a further understanding from the intricacies of developing with the Arma 3 game engine. Following that, a series of tests will then be executed to deduce if there are any problems with the projects code. Finally, a detailed look on the system implementation, including the application of the algorithms chosen in the section 3.

4.1 Chosen Technologies and Supporting Modifications

Developing custom modifications for Arma 3 is a process that can be achieved via the use of numerous different methods and solutions, all of which being tailored to different circumstances and offering distinct advantages and disadvantages.

It was first decided to develop this project using the Arma 3 game engine, as opposed to using other third party technologies. Arma 3 was chosen for its already existing vehicular combat features that will act as a base for this project to build upon. This will save time when compared to developing within another game engine, such as the Unity game engine, as basic vehicle combat features would have to be developed. The software engineer is also experienced in the development of Arma 3, thus cutting down on the initial time spent on familiarisation with the development tools.

To conduct the majority of development it was determined that a C++ extension will be used. Using a C++ extension will offer increased performance, when compared to using in-engine scripting, and a more thoroughly developed IDE, with debugging tools such as breakpoints and performance monitors that are not available within the Arma 3 game engine.

Finally, there will be third party developed modifications in use, one being Community Based Addons (CBA), which will ease and help the development process by providing a number of development tools, most importantly being the extended event handler system. The other modification that will assist in the project development will be example code provided by “KillZone Kid” in the tutorial series on creating an extension for Arma 3. This example code provides a base for setting up a multithreaded application that will alleviate the initial prototyping phase.

4.2 Hardware

Project development and testing was conducted on a custom-built desktop PC with the following specifications:

- Intel® Core™ i5 4670k, 3.4 Ghz, Haswell™ Architecture
- 8GB DDr3 RAM
- Radeon™ RX 470 Graphics card

4.3 System Design

The initial prototype of this projects development involves an interface between an external C++ extension (a .dll library) and several functions created using the in-game scripting language (SQF). Data passed along the interface is of a string type resulting in data parsing facilities on both sides of the interface. The interface is set up using a multi-threaded solution allowing the game engine to resume activates while the C++ extension executes its algorithms.

The “HitPart” function is created upon detecting an impact of any vehicle using the extended event handler system via the use of CBA. The “HitPart” function packages the vehicle and projectile properties into a structured string with delimiters; then sending them to the C++ extension through an asynchronous function. The asynchronous function uses a ticket system where it can send multiple tickets with different IDs and parameters for sending, receiving and checking for results. The extension uses a threaded worker method to execute the tickets consecutively by applying the algorithms discussed in section 3.4 and 3.5. Once the ticket has been processed it is then sent back to the asynchronous function. Upon receiving the ticket, the data is parsed and the post-impact variables of the projectile and target vehicle are applied, all executed within functions created within the Arma 3 game engine.

4.4 User Interface Design

By the nature of this project there has been no user interface designing taking place. However, it is important to explain the user interface of vehicle combat within Arma 3 as it shows damages to the vehicle currently in use. This relates to this project as damage changes of the modules of a vehicle will show up in the user interface within Arma 3.

As seen in the diagram above, there are small bars with text representing the different modules of a vehicle. Each bar changes colour from white, to orange and finally to red as the hit points of each vehicle decreases. There are some modules that this project implements damage for that the current user interface does not show, however it would be time-consuming to accommodate for these modules by modifying the default user interface, not to mention that it would be beside the point of the projects original aims and objectives.

4.5 Experimental Design

As Arma 3 is currently lacking in documentation, a series of experimental tests will be developed to acquire a further understanding of the game engine and how it will integrate with the projects development. To conduct said experiments, the following tests have been designed:

1. Investigate if the projectile / target properties can be modified during and after impact.
2. Determine the effectiveness of a C++ extension and if it can be used to change the projectile / target properties.
3. Examine the projectile / target properties retrieved and if they are accurate enough to be used for the development of the project.

4.5.1 Experimental Test 1 - Investigate if the projectile / target properties can be modified during and after impact

This test is integral to the success of the project, as without being able to change the properties of the projectile, more specifically the vector, velocity and the damage done to the target, this project will be impossible to complete.

To setup this experiment a test environment will be created within the game with a tank firing a projectile at another armoured vehicle. Using the in-game scripting language and the popular mod CBA (Community Based Add-ons), an event handler will fire when the target tank is hit. This event handler will then return the projectile properties which will be used to change the projectile flight path and the damage done to the vehicle. The projectile flight properties will be recorded to a text file at set time intervals to determine if the experiment has been a success.

4.5.2 Experimental Test 2 - Determine the effectiveness of C++ extension and if it can be used to change the projectile / target properties

Using a C++ extension to conduct the majority of development will give a multitude of advantages. Being able to pass data to and from a C++ extension will bring better performance and a more efficient development environment.

This experiment will test whether data can be transferred between a C++ extension and the Arma 3 engine. It will also investigate what the form the data passed is in and the performance of the transitions. Performance is of particular importance because of the rapidity of time with projectile ballistics.

To setup this experiment, the test environment created for experimental test 1 will be used and built upon. A C++ extension will be integrated within the test environment and be used to change the projectile and target vehicle properties once a shell has impacted the vehicle. The changes made will be obvious in nature allowing the assessor to see at first glance if the test had been a success or not. In detail, these changes will be to set the target vehicle health to 0 after impact, consequently destroying the vehicle, and modifying the velocity of the projectile to reflect at a ninety-degree angle from the initial impact line.

4.5.3 Experimental Test 3 – Examine the projectile / target properties retrieved and if they are accurate enough to be used for the development of the project.

There are various properties of a projectile / target when impact has occurred, most notably are the surface normal vector and position at the point of collision between projectile and its target. The format of both properties will be investigated further. If development proceeds without knowing these values, then any calculations that occur will not be valid.

To conduct this test, a mission will be created based upon experimental test 1. Once the projectile is fired, a third-party script will be used to record the projectile flight path via the drawing of visible lines. The position will be recorded and then a spherical test object will be placed at the recorded coordinates. Further tests will be then conducted upon the test object to determine the format of the position property. The surface normal vector will then be recorded and the projectiles flight path will be modified to this value, clearly showing the surface normal direction.

4.6 Test Design and System Testing

For this project, a number of tests will be designed to confirm that the coding is working and correct and meets with the project aims and objectives. The most notable area of development that can be tested is related to objective 4 and 5. To further explain, the calculations done when a projectile has impacted a target and whether the projectile will penetrate along with the post-impact properties of the projectile / target.

To conduct this test, data will be gathered from external research done on the penetration ability of select projectiles. This research involves data gathered from World War 2 sources including live firing tests and their results. Noted down during these tests are the type of projectile fired, range of projectile flight, thickness and type of target material impacted and the penetration depth of the projectile. Once assembled, the equation chosen, to complete objective 4 and 5, will use the data gathered and the outcome will be analysed and compared against the external research results.

Tests will be conducted to determine the accuracy and what is exactly happening with the module damage system, all in relation to objective 6. To conduct this test a series of fire tests will occur where the projectiles position will be recorded. The fire tests will be of a varying nature to include a significant amount of the probable situations that can occur during normal Arma 3 vehicle gameplay.

4.7 System Implementation

This section will explain in detail some of the more complex areas of this project that are not obvious to what they are achieving at first glance. The areas that will be described within this section will be the event handler system, the ticket system used to set up the interface between the external library and the Arma 3 game engine and finally, the multithreaded solution used within the external library.

4.7.1 Event Handler System

An event handler is a system that allows a user to automatically monitor the Arma 3 game engine and execute custom code when a certain event handler's requirements are met. For example, the "HitPart" event handler, the event handler used for development of this project, will fire upon detecting an impact between a projectile fired from a shooter and any vehicle within the game. Native use of the event handler system requires the event handler to be attached to the relevant object within the game. In the case of the earlier example, this would require the "HitPart" event handler to be attached to every unit in the game that has the means to fire projectiles, namely every human soldier within the game.

Attaching the "HitPart" event handler to every unit within a particular mission would require a bloated solution that would be prone to bugs and performance issues. Not to mention the exponential increase on development time that this solution would incur. Luckily however, there is an answer to this problem in the third party developed modification "Community Based Addons". Community Based Addons is a modification with a multitude of developer tools that can assist with the development of modifications for Arma 3. The developer tools from Community Based Addons that have been used in the development of this project is the extended event handler system. This system allows a developer to use an event handler without attaching it to a relevant unit; each time the requirements are met for a particular event handler it will execute the custom code laid out beforehand. This offers numerous advantages to using the base Arma 3 system, allowing a solution less prone to bugs because the software engineer need not worry about designing a solution to attaching an event handler to the relevant units.

The event handler that has been used in relation to the development of this project is the “HitPart” event handler. This event handler will run when an object gets injured/damaged. Subsequently, a custom function will execute, outputting the variables contained within the event handler into separate types. The data contained within the “HitPart” event handler consists of the following:

- Target: Type, Object – Object that got injured/damaged
- Shooter: Type, Object – Unit or vehicle that infliction the damage
- Bullet: Type, Object – Object that was fired
- Position: Type, Position3D – Position the bullet impacted (Above Sea Level)
- Velocity: Type, Vector3D – 3D speed at which the bullet impacted
- Selection: Type, Array – Array of Strings with named selection of the object that was hit
- Ammo: Type, Array – Ammo info: [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name] OR, if there is no shot object [impulse value on object collided with, 0,0,0]
- Direction: Type, Vector3D – Vector that is orthogonal to the surface struck
- Radius: Type, Number – Radius (size) of component hit
- Surface: Type, String – Surface type struck
- Direct: Type, Boolean – True if object directly hit, false if hit by indirect damage

As seen above, the “HitPart” event handler provides all the necessary variables that will be used within the penetration calculations of this project. The most notable variables that have been essential in the penetration calculations are the “Target”, “Bullet”, “Position”, “Velocity”, “Ammo” and “Direction” variables.

4.7.2 Ticket System

To set up and pass data along the interface a multithreaded solution has been developed to increase performance and to not slow down the game while the external library executes its code. This system has been developed from example code provided by “KillZoneKid” during his Arma 3 C++ extension tutorial.

This system uses an asynchronous function to send and retrieve data from an external library of the C++ language. The data type passed along this interface is a single string variable, therefore requiring string parsing facilities on both sides of the interface. The asynchronous function allows the Arma 3 game engine to resume its activities, only re-joining the main Arma 3 thread upon completion in the executing of its code.

The data passed into the external library could be done by simply sending one request at a time. However, this would be slow in comparison to allowing the external library itself to operate on another thread. The ticket system used within this project allows external library to operate on another thread and being able to manage multiple requests at a time. This is done by first developing the external library to contain a ticket stack. A worker function can then go through each ticket in the stack consecutively.

To manage the tickets, a number of commands have been set up allowing the external library to communicate with the asynchronous ticket function within the Arma 3 game engine. These commands are split into two sections, first, the commands used by the ticket function in regards to the sending and retrieval of tickers and secondly, the commands used by the external library in the responses to the earlier mentioned commands. These commands are the following:

- “s:string” – The ticket function calls the extension by sending data along the interface, where “s” is the send request and “string” is string variable sent. Upon receiving a ticket, the extension will respond with the ticket number.
- “r:ticketnumber” – The ticket function can query whether a ticket is ready, where “r” is the send request and “ticketnumber” is the relevant ticket number. The extension will then respond with either of the three commands listed below after receiving the ticket request.
- “WAIT” – indicates that the ticket in question is not ready to be retrieved yet.
- “EMPTY” – The ticket request is empty and does not exist.
- “RESULT” – The ticket is ready and the extension will respond with the result of its execution, “RESULT” is the custom string data, defined by code within the extension.

4.7.3 Multithreaded Solution within the C++ Extension

To increase performance and reduce possible bugs a multithreaded solution was developed for the C++ extension. This solution closely interlinks with the ticket system described above in section 4.5.2. This system was based and expanded upon the example code given by “KillZoneKid” in his Arma 3 C++ extension tutorial.

As stated in section 4.5.2, a ticket system is in place to handle all extension calls. The extension contains a ticket stack of unordered map data type, with a long int containing the ticket Id and a data struct with the relevant variables used within the algorithms of the extension. To handle the execution of the tickets a worker thread has been set up to sequentially carry out the algorithms within the extension using the ticket data.

Using a multithreaded solution does not come with its downfalls however, as care has to be taken to ensure globally available data is not modified by more than one thread at the same time. To achieve this, a mutex type is implemented to “lock” and allow safe modification of global data by a particular thread, before “unlocking” the data again. Using the “Mutex.Lock()” command only allows the owning thread to alter the data that has been locked, after which the “Mutext.Unlock()” command is called, undoing the lock and allowing other threads to access that data again.

5 Evaluation

This section will provide a final evaluation of the completed developed done up to this point. First, a list of what this project has achieved in the given time frame and how this compares to the original vision. To expand upon that, a list will be laid out of the initial objectives set in section 2 and compared to the developed code to deduce whether these objectives have been met. This section will ask whether the objectives have been completed partially, fully or not at all. Notes will be made on the completion of these objectives such as what could have been done better, was the initial objective laid out correctly or could it be rewritten to be more relevant to the vision of this project? Finally, any further work to this projects development that could be done will be questioned.

5.1 Project Achievements

Overall, considering the early troubles founded in the initial development stage of this project, the work completed so far have been to a good standard, even exceeding what was expected from the initial objectives as the tasks laid out in section 2 had to be expanded upon as they were all completed around the halfway point in this projects development cycle. The new objectives theorised after the halfway point were objective 7 and 8. These objectives involved the expansion of the current penetration equation in their accuracy and the range of projectile types they covered. This involved further research done into the mechanics of armour penetration by projectiles to acquire the algorithms discussed in section 3.5 that would more accurately simulate the dynamics of modern vehicular combat.

The task at hand was initially anticipated to be significantly less complex, time consuming and challenging than what was discovered after committing to proper development of this project. This is due to the substantial amount of learning, through documentation and experimental testing, that the software engineer had to undertake. This learning involved the setting up of an external C++ library with Arma 3, this task seemed simple at first yet provided a number of initial obstacles that proved time consuming to overcome. These obstacles proved to be general debugging of the C++ extension, most notably how the code written could not be paused and stepped through one line at a time, instead the code had to be debugged manually through the use of if statements and commenting out possible code causing the project to break. As can be envisioned from using such a process, this significantly increased the projects development time. The software engineer also had to learn an entirely new scripting language, namely SQF. SQF is not unlike the more popular object oriented computer languages such as C++, however it does have a few particulars that proved hard in adjusting to. Finally, the last major hindrance in this project development was the limitations of the Arma 3 game engine, more notably the lack of useful scripting commands in relation to the modification of vehicle combat. The more prominent of these were the retrieval of any armour thickness value at a specific location on a vehicles surface, the complete lack of projectile simulation within the vehicle itself (this caused a significant jump in development time as a large, bloated workaround system had to be developed) and lack of any mathematical functions within the SQF scripting engine, contributing to the use of a C++ extension for further development of this project.

At first glance, it may seem that this project did not achieve much and the development done could be considered to be rather lacklustre. However, it is important to note that a substantial portion of the development process was spent on the learning of the SQF scripting language and developing with Arma 3, research into the real-world mechanics of armour penetration by large calibre projectiles and finally, finding the relevant equations that were used within this project.

5.1.1 Objective 1 – Retrieve from the game engine the properties of a projectile and its target after impact – Completed.

This objective was completed in a fairly short time frame, with only a few shortcomings and hindrances. These hindrances involved the setting up of the event handler system, as initially the Community Based Addons modification had not been discovered and it proved problematic to attach an event handler to every unit within a particular scenario within Arma 3. However, this was overcome from using the extended event handler system that the Community Based Addons mod provides.

5.1.2 Objective 2 - Interface with an external library and pass data between said external library and the game engine – Completed.

This objective upon its theorisation looked to be reasonably simple to complete. However, this initial idea proved to be the complete opposite in practice as there were numerous setbacks and obstacles involved in setting up an interface between the Arma 3 game engine and an external library. The most apparent of these were the complete lack of any debugging tools provided by Arma 3, only a simple script error with a simple description of the error and its location is present (even this system often proved to give wrong and misleading information). As evident, this produced a substantial increase in the development and completion of this particular objective.

5.1.3 Objective 3 – Parse the data outputted into an external library and store as variable for use within the library – Completed.

This was another objective that at first glance, seemed to be relatively straightforward to implement. Upon committing to actual development, this objective proved to be time consuming and incredibly difficult to complete. As discovered before in the development of objective 2, there is a lack of useful debugging tools when developing with Arma 3. The most notable bug that could have been found via the use of said tools was the parsing of a long string variable into separate lines, interestingly what was found was that the "stringstream" class and its relevant methods proved to not work at all when used within an external library of the Arma 3 game engine. To workaround this simpler, yet longer in length, parsing systems were used.

5.1.4 Objective 4 - Calculate the penetrating ability of the projectile after impact and update the game engine appropriately – Completed.

Interestingly enough, also contributing to the fact that developing with the Arma 3 game engine proves to be difficult to predict the complexity of developing particular systems, this objective was completed without any major interruptions. In fact, the most time-consuming part to implementing this objectives criterion was the research and finding of the appropriate penetration equation, due to a lack of academic research into the mechanics of armour penetration by large-calibre projectiles.

Most research that was discovered was too complex for the original vision of this project or was not appropriate for the vehicular combat situations that this project is developed for.

5.1.5 Objective 5 - Split armour thickness values of the target vehicle based on the position of impact – Completed.

After the trialling of multiple algorithms to complete this objective and the design of the system that ended up being developed, this objective proved to be yet again simple to complete. Due to the limitations of the Arma 3 game engine it was decided to develop a workaround with the C++ extension. The system in place is quite accurate, even accommodating for more complex damage models to be possibly developed in the future.

5.1.6 Objective 6 - Create a module-based damage system where individual modules of the target vehicle can be damaged – Completed, but with bugs.

This system was implemented in a similar fashion to that of objective 6. However, problems began to show upon the testing phase of this project. Due to the difficulty in debugging with Arma 3, it would prove to be incredibly time consuming to develop to a point with no system breaking bugs, albeit these bugs are rare in their occurrence.

5.1.7 Objective 7 – Further develop the penetration calculation to account for HEAT (High-Explosive-Anti-Tank) shells – Partially Completed.

This was the first of two objectives theorised after the halfway point in this projects development lifecycle. This objectives equation was simple and straightforward to implement. However, problems arose in the implementation on the damage that a HEAT shell would invoke upon a target vehicle, therefore only being partially completed due to time constraints.

5.1.8 Objective 8 – Further develop the penetration equation to incorporate factors important to developing an accurate simulation – Partially Completed.

This objective was the second of two set out after it was found that the development of this project was ahead of schedule and plans were created to expanded the ballistics simulation that this project provides. This objective was time consuming to implement, only due to the length and complexity of the relevant equation however, this objective was developed with no major stoppages to the development process. In light of this, only partial testing was conducted due to time constraints invoking numerous possible errors.

5.2 Further Work

If more time were available in the development of this project there are numerous features and improvements that could be implemented to further increase the level of simulation of vehicular combat that this project provides. These features could include, amongst others:

- The simulation developed by this project could be expanded to include multiplayer scenarios, this would require further research into multiplayer development within Arma 3. However, if this project were to be released as a publicly available modification, this would almost be an expectation and requirement.
- Within Arma 3, not unlike the vehicular combat simulation, there is a significant lack of proper penetration calculations and simulation of the dynamics of missile penetration of armoured targets. This system could prove to be complex and time consuming to implement, as entirely new research into the field would have to be conducted.
- Expand the current system to include all vehicles within Arma 3, even unarmoured light vehicles such as cars. As this project has only focused on a limited number of vehicles it would be essential for public release to include all vehicles in the vanilla game of Arma 3 into this projects simulation. Expanded upon this, an automatic system could be developed to include any third party developed modification that contains any new armoured vehicles.

In addition to the features stated above, further development time could be contributed to increasing the solidity of this projects code and the reducing in the number of possible bugs.

6 Conclusion

The initial vision for this project was not to provide an exact simulation of the real-world mechanics of armour penetration by large calibre projectiles, but instead to increase the level of simulation of vehicular combat within Arma 3 and bring it up to the level that the rest of game sets, and, up to the expectations from the community surrounding Arma 3.

In regards to the initial vision, this project has achieved that and more. It has provided advanced penetration equations and complex damage models accounting for armour thickness locations and even allowing damage models for internal components. This project does not come without its shortcomings however, as there are numerous bugs and errors that could have been fixed if the testing had been designed to be more thorough and efficient in its detection of possible errors with this projects code.

References

Achim Sippel, H.-J. K. (1991). *United States of America Patent No. US5009167 A.*

Ballistic Analysis Laboratory . (1968). *PROJECT THOR TECHNICAL REPORT NO. 47 .* Baltimore: Ordnance Research and Development Project No. TBJ-0238 .

Bohemia Interactive. (2013). Retrieved from Arma 3: https://arma3.com

Bohemia Interactive. (2017). *Arma 3 Tanks Config Guidelines*. Retrieved from Community BI Studio: https://community.bistudio.com/wiki/Arma_3_Tanks_Config_Guidelines#Armour_plates_setup

Bohemia Interactive. (2017). *Arma 3; Event Handlers*. Retrieved from Community BI Studio: https://community.bistudio.com/wiki/Arma_3:_Event_Handlers

Brothers, J. (1968). *United States of America Patent No. US3416449 A.*

Buc, S. M. (1994). *United States of America Patent No. US5297492 A.*

Captain Jonathan M. House, U. A. (1968). *Toward Combined Arms Warfare:- A Survey of 20th~Century Tactics, Doctrine, and Organization.* Washington D.C: U.S Goverment Printing Office.

Franze, Waffle SS, & Tom 48 97. (2015). *LOD*. Retrieved from Community BI Studio: https://community.bistudio.com/wiki/LOD#Fire Geometry

House, J. M. (1984). Toward Combined Arms Warfare.

John S. Foster, J. A. (1983). *United States of America Patent No. US4418622 A.*

Lanz, W., & Odermatt, W. (1992). Penetration Limits of Conventional Large Calibre Anti Tank Guns / Kinetic Energy Projectiles. *13th International Symposium on Ballistics*.

Lorrin Rexford Bird, R. D. (2001). *World War II Ballistics, Armor and Gunnery.* New York: Overmatch Press.

Olds, T. 4. (2015). *Arma 3 Damage System*. Retrieved from Community BI Studio: https://community.bistudio.com/wiki/Arma_3_Damage_Description

Thomanek, F. R. (1978). *United States of America Patent No. US4111126 A.*