FELADATKIÍRÁS

A mai világban elterjedtek a lánctalpas, nagy tömegű és teherbírású járművek, ezek közül az egyik fontos kategória a harci célt szolgáló eszközök. Ezek virtuális modellezése kifejezetten érdekes lehet, hiszen nem megfelelő terhelhetőség, védelem vagy tűzerő esetén a valós példányok gyártása nagy veszteséget termelhet. A szakdolgozat feladata ezen gépek valószerű modellezése számítógépes játékokhoz a Unity játékmotorban, néhány kiemelt kategóriára fókuszálva. Az egyik kategória a felfüggesztés és a lánctalp működésének szimulálása. Ez magában foglalja a jármű meghajtását, a lánctalpak valósághű reprezentációjával, valamint a görgők többféle terepen történő, attól függő rugózásának megjelenítését. Szimulálandó továbbá a löveg által kilőtt lövedékek hatása páncéllemezek ellen, eltérő szituációkban. A legtöbb járműhöz sokféle lövedéktípus létezik, melyek közül mindegyiknek vannak előnyei és hátrányai, mely meghatározhatja, hogy mikor melyik típussal érdemes tüzelni. A lövedékek becsapódásának helyén emellett a páncél esetleges deformációit, sérüléseit is modellezni kell, mely alapján a jármű épsége és harcképessége meghatározható. A szakdolgozat feladatai a következők:

- Tekintse át lánctalpas járművek főbb jellemzőit leíró elméleti hátteret.
- Tekintse át Unity játékmotort, elsősorban a fizikai alrendszerre, és annak a jelen szakdolgozathoz felhasználható komponenseire fókuszálva.
- Tervezzen meg egy rendszert mely valószerűen modellezi a lánctalpas páncélos járművek fizikáját, beleértve a lánctalpak mozgását, a különböző lövedéktípusok viselkedését és azok becsapódásának hatását a páncélzatra. Tervezzen meg egy egyszerű demonstrációs játékot a fentiek bemutatására.
- Implementálja a rendszert és a demonstrációs játékot a Unity játékmotorban.

Értékelje az elkészült rendszert valószerűség és hatékonyság szempontjából.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Pető Ádám

PÁNCÉLOS JÁRMŰVEK FIZIKAI MODELLEZÉSE SZÁMÍTÓGÉPES JÁTÉKOKHOZ

KONZULENS

Dr. Magdics Milán

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Pető Ádám**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 10.

...…….
Pető Ádám

Összefoglaló

A szakdolgozatban létrehoztam egy páncélos jármű széleskörű, valóságszerű virtuális modelljét a Unity játékmotor segítségével.

A felfüggesztés az igazihoz nagyon hasonlító módon lett megvalósítva, amely kiterjed többek között arra is, hogy a görgők felveszik a talaj adottságait, valamint a felfüggesztés rugójának erejét a hidropneumatikus felfüggesztéshez hasonlóan lehet manuálisan állítani.

A felfüggesztéshez csatolt lánctalp fizikai modellezése lehetővé teszi, hogy a teljes lánctalpat a hátsó görgő segítségével hajtsuk meg, de a lánctalpelemek egymáshoz csatlakoztatása is az igazihoz hasonlóan lett megvalósítva.

A projekt többféle lövedéktípust is képes szimulálni, melyek működésükben sok szempontból eltérnek egymástól. A modell emellett képes meghatározni, hogy egy adott lövedék át fog-e ütni egy adott páncéllemezt, adott körülmények között.

A projekt továbbá szimulálja az egyes lövedékek becsapódásának hatásait, melynek megvalósításához képes a modellek háromszöghálóját megfelelő módon sűrűsíteni, annak csúcsainak pozícióját a becsapódás paramétereinek függvényében megváltoztatni.

Ezek bemutatásához létrehoztam egy demonstrációs játékot, mely implementálja az összes korábban leírt funkciót, és szemlélteti ezek működését.

Abstract

In my dissertation, I created an extensive, realistic virtual model of an armored vehicle using the Unity game engine.

The suspension has been implemented in a very similar way to the real one, which includes the fact that the rollers absorb the properties of the ground, and the force of the suspension spring can be adjusted manually, like in the case of hydropneumatic suspensions.

The physical modeling of the track attached to the suspension allows the entire track to be rotated by the rear roller, but the connection of the track elements to each other has also been implemented in a similar way to the real one.

The project can simulate several types of projectiles, which differ in many respects in their operation. The model is also able to determine whether a particular projectile will penetrate a particular armor plate under given conditions.

Furthermore, the project simulates the effects of the impact of each projectile, for which it can properly densen the triangular mesh of the models and change the position of its vertices as a function of the impact parameters.

To display these, I created a demonstration game that implements all of the features described earlier and illustrates how they work.

1 Bevezetés

A mai világban elterjedtek a lánctalpas, nagy tömegű és teherbírású járművek, melyek az ipar több területén is felhasználásra kerülnek. Többek közt fontos szerepet látnak el alapvetően nehezen mozdítható tárgyak, nagy súlyú törmelékek mozgatásában, akár kifejezetten durva terepen is, például építkezéseken, ahol általában több ilyen járművel is találkozhatunk, ilyenek a daruk és a markolók. Ezek mellett a hadi technológia is számottevő mennyiségben használ ilyen járműveket, de itt nem nagy tömegű külső súlyok mozgatása a cél, mint a korábban felsorolt gépek esetén, hanem a tömeg nagy részben az jármű védelmét, páncélzatát teszi ki. Ez a diplomamunka alapvetően az utóbbi felhasználási móddal foglalkozik, azon belül pedig ezeknek a berendezéseknek a modellezésével.

1.1 A feladat értelmezése

A feladat tulajdonképpen ezen modellezés valószerű megvalósítása számítógépes játékokhoz a Unity játékmotor segítségével, néhány kiemelt kategóriára fókuszálva, melyek:

- A felfüggesztés és a lánctalp szimulálása
- Az eltérő lövedéktípusok megvalósítása
- A löveg által kilőtt lövedékek hatása a páncéllemezek ellen

1.2 A tervezés célja

A tervezés alapvetően többcélú. Egyrészt lehetőséget biztosít számítógépes játékokban arra, hogy ilyen járműveket vezessünk, ezekkel harcoljunk, valósághű körülmények között, ezzel egy magával ragadó játékélményt nyújtva a játékosok számára. Másrészt pedig a valószerű virtuális modellezés lehetővé teszi leendő valós példányok széleskörű tesztelését, tervezési hibák felfedezését anélkül, hogy ezekből valódi prototípust kelljen gyártani, amely gazdaságilag mindenféleképpen jövedelmező befektetés. Ezek mellett pedig kisebb módosítások tesztelése is lehetséges, mellyel könnyedén lehet finomhangolni a jármű karakterisztikáit, így maximalizálva annak tűzerejét, védelmét és mozgékonyságát.

1.3 A diplomaterv felépítése

A diplomaterv felépítése alapvetően 3+2 elemi részre tagolható. A három elsőrendű rész a feladat értelmezésében felsorolt három pont kifejtése, ezek megvalósításának potenciális módjainak ismertetése, a legmegfelelőbb kiválasztásának részletezése és magyarázata. A további két rész közül az egyik a feladatkiírás pontosítása és részletes elemzése, valamint egy ismertető a páncélos harci járművek működéséről, a másik pedig a különböző feladatrészek együttműködésének vizsgálata és az elkészült projekt összefoglaló elemzése.

2 A feladat részletes leírása

Az alábbiakban a páncélos harci járművek működésének részletes leírását szeretném ismertetni, főleg a feladatkiírásban ismertetett kategóriákra fókuszálva, valamint a feladatkiírást szeretném mélyebben kifejteni, kitérve már pontosabb specifikációkra is, melynek segítségével pontosabb képet kaphatunk a készítendő projekt pontos jellemzőivel kapcsolatban.

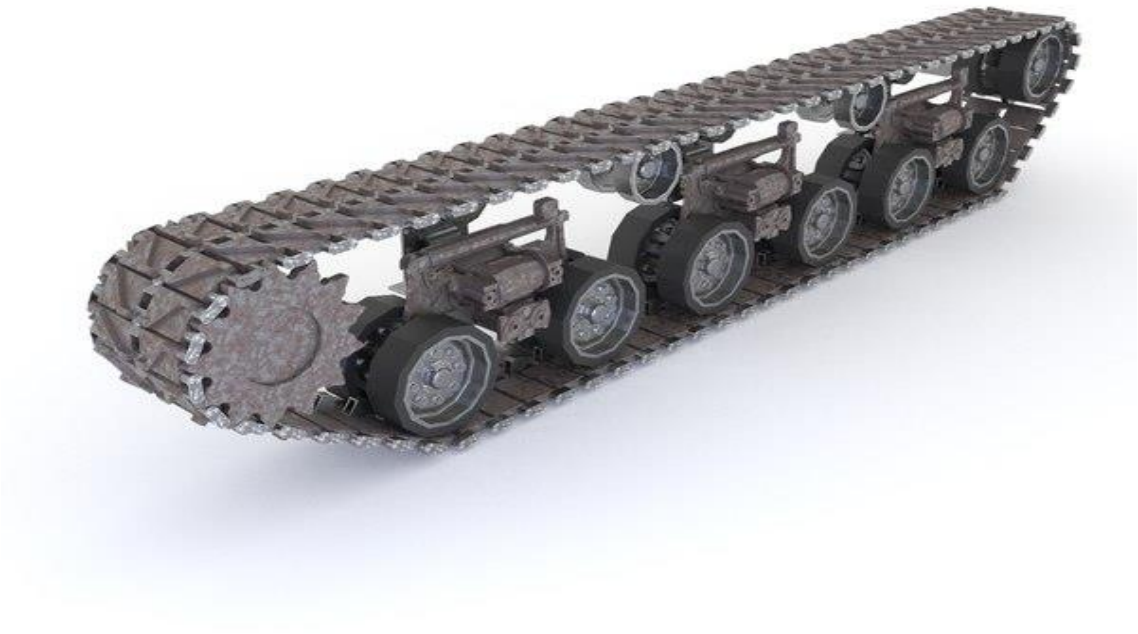
2.1 Alapkoncepció

Alapvetően egy M1 Abrams típusú, amerikai MBT (Main Battle Tank) harckocsi (2.1. ábra) modellezését valósítja meg a Unity motor segítségével készített demonstrációs projektem.

2.1. ábra: M1 Abrams

2.2 A felfüggesztés és a lánctalp

A lánctalpas járművek általában két lánctalpsorral rendelkeznek, egyik a jobb, a másik pedig a bal oldalon található, végig a gép hossza mentén. A 2.2 ábrán egy lánctalp kialakítását láthatjuk.



2.2. ábra: Példa lánctalpra és felfüggesztésre

Ezek egyik végén található a meghajtásért felelős görgő, amely a motortól kapott erőt adja át a lánctalpnak, tulajdonképpen ez felelős a lánctalp forgatásáért. A legtöbb görgő viszont a lánctalp aljával érintkezik, ezek osztják el a gép terhét maguk között. Ezeket általában egyesével vagy párosával szokták valamilyen rugalmas módon felfüggeszteni, ezzel lehetővé téve, hogy fel tudják venni a terep adottságait, így mindenféle terepen képesek elosztani a terhet. Található még egy görgő a jármű elején is, hasonló magasságban a hátsó görgőhöz. Ezek a görgők azért vannak kissé megemelve a többihez képest, hogy a jármű képes legyen könnyebben áthaladni kiálló akadályokon, valamint lehetőséget biztosít a gép előre-hátra billegésére is – a felfüggesztés által biztosított kereteken belül – így ezeket a kerekeket nem kell rugózó felfüggesztéshez

csatolni. Szokás néha még kisebb görgőket tenni a lánctalp felső részének megtámasztásához is, de ezek nem mindig találhatóak meg.

2.2.1 Hasonlóságok és különbségek az autók kereke és a lánctalp között

A felfüggesztés és a lánctalp együttese sok szempontból hasonlít az autókon használt kerekekhez, de rengeteg különbséget is felfedezhetünk a két konstrukció között. A legszembetűnőbb talán az, hogy a lánctalpas járművek képesek egyhelyben is megfordulni. Alapvetően utakon nem ok nélkül használunk autókat, kerekes járműveket. Utakon a kerekes járművek gördülési ellenállása nagyjából fele akkora, mintha lánctalppal szerelnénk fel őket, így kevesebb üzemanyagot igényelnek. Terepen viszont más a helyzet. Itt egy kerekes jármű legalább annyi üzemanyagot fogyaszt, mintha lánctalpas lenne. Továbbá terepen a megkapaszkodás csak egy adott besüllyedésig lehetséges, ebben nyújt nagy segítséget a lánctalp, amely megakadályozza, hogy egy-egy görgő túlságosan mélyre essen be, ezzel mozgásképtelenné téve a járművet.

2.2.2 Felfüggesztés

Az Abrams esetén a felfüggesztés működése több szempontból is érdekes. Szerettem volna minél több lehetséges funkciót megvalósítani, így végül is - habár csak egy ilyen prototípus készült – úgy döntöttem, hogy hidropneumatikus felfüggesztéssel fogom felszerelni a modellt. [1] A prototípus a 2.2.2. ábrán látható.

2.2.2. ábra: Hidropneumatikus felfüggesztéssel rendelkező Abrams prototípus

2.2.2.1 A felfüggesztés módja

A tank felfüggesztése csak néhány szempontból tér el a 2.2 ábrán láthatótól. A legfontosabb az, hogy minden egyes görgő külön-külön van hozzácsatolva a jármű

testéhez, alapesetben torziós rudakkal, jelen példában hidropneumatikus felfüggesztéssel. A meghajtást a hátsó görgő adja.

2.2.2.2 Hidropneumatikus felfüggesztés

A tankokhoz sokféle felfüggesztési módot, típust használnak, például a Christie-felfüggesztés, a függőleges csavarrugós felfüggesztés, vagy a torziós rudas felfüggesztés, de ezek nem képesek a rugóerejük változtatására. Ezzel ellentétben a hidropneumatikus felfüggesztés lehetőséget biztosít erre, így a jármű teste tulajdonképpen egy bizonyos fokig szabadon mozgatható a lánctalpak felett. [2] Ez sok szempontból előnyt adhat a járműnek, például ha előre döntjük a testet, akkor a löveg képes még jobban lefelé mutatni, ezzel lehetővé téve, hogy meredekebb dombok mögül is minimálisan kelljen a jármű testét mutatni, így kisebb célponttá téve önmagát.

Jelen projektben cél ennek a viselkedésnek a valóságszerű modellezése.

2.2.3 Lánctalp

A lánctalp alapvetően sok egymáshoz kapcsolt fémlap összessége, amelyek a görgők körül forognak. Ezek érintkezek a talajjal és adják meg a jármű tapadását. A kapcsolatok lehetőséget nyújtanak a lánctalp elemek közt forgásra egy tengely körül, így lehetővé téve, hogy a lánctalp felvehesse a terep adottságait.

Ezt a viselkedést valósítja meg a projekt is. Tehát a járművek meghajtása ott is a lánctalpakon keresztül történik.

2.3 A kilőtt lövedékek hatása a páncéllemezekre

Alapvetően a kilőtt lövedékek lövedéktípustól függő módon képesek átütni a páncéllemezt, nem átütni a pácéllemezt, de gellert is kaphatnak, ha megfelelő szögben érkeznek a lemez felé. Amennyiben egy lövedék gellert kap, akkor csak kis mennyiségben gyengíti a páncélt, az átütő és nagy átütéssel rendelkező, de nem átütő találatokhoz képest. Így a modellben a gellert kapó lövedékek nem változtatják meg a páncél felépítését, formáját. A másik két esetben viszont igen. Ezek a becsapódások képesek szemmel látható módon behorpasztani a páncéllemezeket. A deformáció mértéke függ az adott lemez vastagságától, és a becsapódó lövedék sebességétől, tömegétől.

Ezeknek a deformációknak realisztikusan kell kinézniük, de mindemellett fontos, hogy a program futásának sebességére ne legyenek nagy hatással.

2.4 Lövedéktípusok

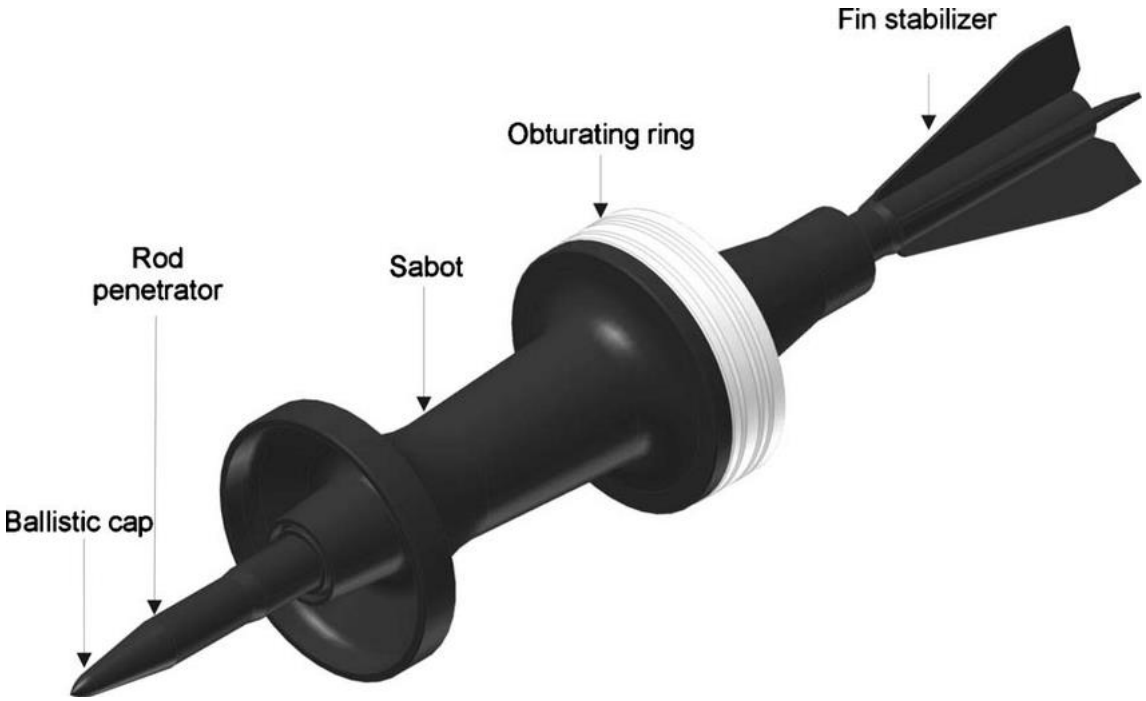
A legtöbb löveg alkalmas többféle lövedék tüzelésére is, melyek mind más-más célt szolgálnak. A két legjellemzőbb lövedékfajta nagymértékben eltérő módon próbálja átütni a páncélt.

2.4.1 Kinetikus energiával működő lövedékek

A kinetikus energiával működő lövedékek mozgási energiájukat használják fel az ellenséges páncél átütésére, a célpontban történő kár okozására. A lövedékek tömege, típusa és keménysége állandó, viszont sebessége a megtett távolsággal arányosan csökken, így átütőképessége is csökken ennek függvényében.

2.4.1.1 APDS és APFSDS lövedékek

Ezek a legerősebb kinetikus energiával működő lövedéktípusok. Az APFSDS (Armour-Piercing, Fin-Stabilized, Discarding Sabot) lövedék (2.4.1.1. ábra) tulajdonképpen egy továbbfejlesztése az APDS (Armour-Piercing, Discarding Sabot) lövedékeknek. Ezek a lövedékek repülnek a legnagyobb sebességgel, valamint a többi ilyen fajta lövedékhez képest jobban viselkednek vastag és döntött páncéllemezek ellen is. [3] A modellben ezen okok miatt ilyen típusú lövedék használata mellett döntöttem.



2.4.1.1. ábra APFSDS lövedék modellje

2.4.2 High-Explosive Anti-Tank lövedékek

A HEAT lövedékek nem a kinetikus energiájukkal, hanem a bennük elhelyezett robbanékony töltetek segítségével okoznak sérülést a páncéllemezekben, melyek robbanása a lövedék elejének irányába fókuszálódik becsapódáskor. Ennek megvannak az előnyei és hátrányai is. Egyrészt a lövedék nem veszít átütőképességéből nagy távolságok alatt sem, másrészt viszont a gyutacsot nagyon érzékenyre kell beállítani, így a lövedékek a legvékonyabb akadályokban is felrobbannak, ellentétben például az APDS lövedékekkel, amelyek gond nélkül képesek áthaladni rajtuk.

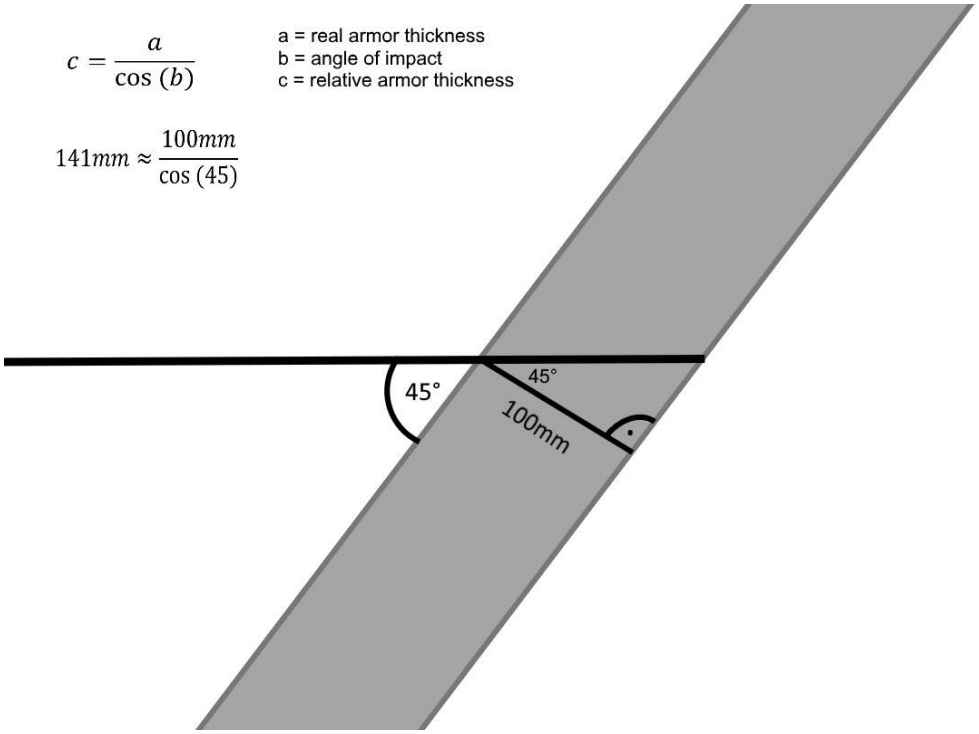
2.4.3 Effektív páncélvastagság

Egy páncéllemez védelmének mértéke nem csak annak vastagságától, de a szembeérkező lövedékkel bezárt szögétől is nagy mértékben függ. [4] Jó példák ennek szemléltetésére a második világháborúban használt német és szovjet tankok.

2.4.3. ábra: Vízközi emlékmű (szovjet T-34 balra, német Pz. IV. jobbra)

Ahogyan a 2.4.3. ábrán is látható, a szovjet tankok nagy szögben döntött felső elülső lemezzel, valamint toronnyal rendelkeztek, míg a németek járművei főleg lapos felületekkel voltak felszerelve. [5] Az effektív páncélvastagság tulajdonképpen a páncél azon hossza, amelyet a lövedéknek meg kell tennie azon keresztül, hogy eljusson a jármű

belső terébe. Van néhány tényező, amely lövedéktípustól függően befolyásolhatja az effektív páncélvastagság mértékét – ezeket a következő pontban fogom kifejteni -, de amennyiben egy lövedék a becsapódás pillanatában egy adott szögben halad a páncél felé, akkor egy trigonometriai képlet felírása után kiszámítható, hogy annak mennyi páncélon kell áthaladnia, hogy átüsse azt.



2.4.3. ábra: Effektív páncélvastagság kiszámítása

A 2.4.3. ábrán egy kétdimenziós reprezentációját láthatjuk az effektív páncélvastagság működésének. Fontos kiemelni, hogy a „b” szög a beesési szög, tehát az a szög, amelyet az érkező lövedék irányvektora, valamint a páncéllemez normálvektora zár be egymással.

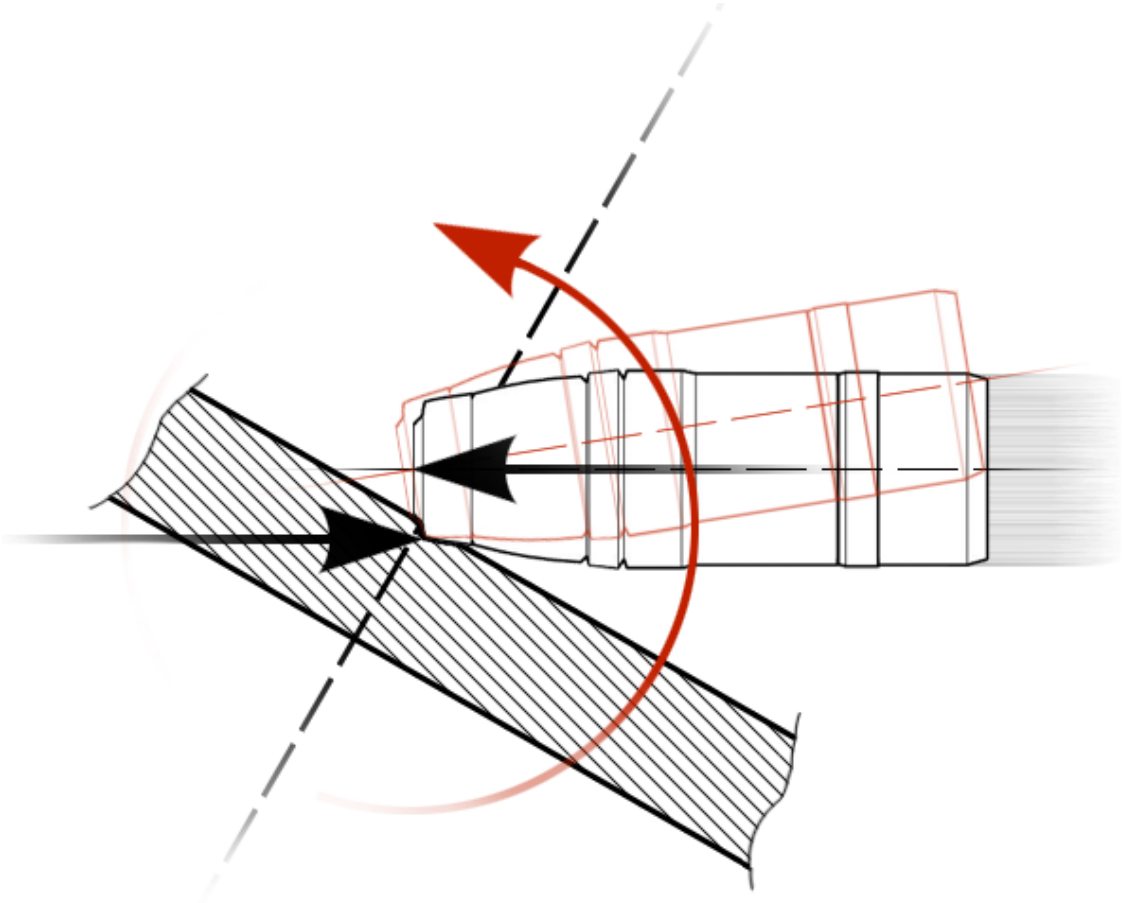
2.4.4 Normalizáció és geller

A legtöbb lövedéktípus rendelkezik két különleges karakterisztikával, melyek sok szempontból befolyásolják azok működését különböző mértékben döntött páncéllemezek ellen.

2.4.4.1 Normalizáció

Vannak olyan lövedéktípusok, jelen esetben a kinetikus energiával működő lövedékek, melyek becsapódáskor változtatnak beesési szögükön kialakításuknak köszönhetően, ezzel csökkentve az effektív átütendő páncélvastagságot. Ennek pontos

fizikai modellezését nem takarja a projekt, röviden csak annyit említenék meg róla, hogy a beesési szög megváltozását a forgatónyomaték okozza. Viszont aránylag pontosan lehet ezt a jelenséget absztrahálni egy normalizációs szög megadásával, amely egy adott lövedéktípust jellemez. A normalizáció működésének kétdimenziós reprezentációját mutatja be a 2.4.4.1. ábra.



2.4.4.1. ábra: Normalizáció szemléltetése

2.4.4.2 Geller

A fénytanhoz hasonlóan a lövedékek esetén is létezik totálreflexió (teljes visszaverődés). Ez akkor következik be amikor a beesési szög meghalad egy kritikus értéket, ilyenkor a lövedék automatikusan gellert kap és aránylag kis mennyiségű sérülést okoz a páncéllemez felületén. [6] Lövedéktípusoktól függően ez a kritikus szög változó.

2.5 Összefoglaló

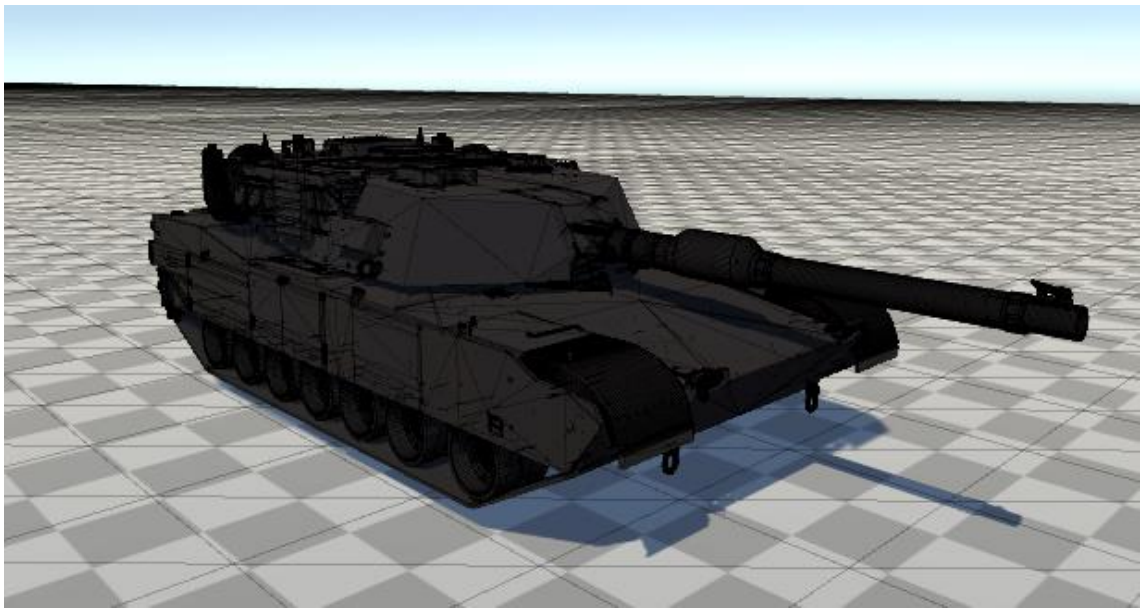
Ezzel sikerült pontosan leírni a feladatot, röviden tehát a projekt célja egy M1 Abrams típusú harckocsi modellezése, kifejezetten kitérve a korábban kifejtett jelenségekre, mechanizumsokra. A továbbiakban már a megvalósítás lehetséges

módjairól, valamint ezek közül az általam kiválasztott megoldásokról lesz szó, melyek alapját az eddig felsorolt elméletek teszik ki.

3 Megvalósítás

3.1 Bevezetés

A projektet Unity játékmotor segítségével valósítottam meg egy Scene keretein belül. A jármű modelljéhez egy, az interneten ingyenesen elérhető modellt vettem alapul, ez a modell a 3.1 ábrán látható. [7]



3.1. ábra: A kiinduláshoz használt 3D modell

3.1.1 Alapvető fizika kialakítása

A projekt első lépése egy gyors vezérlés kialakítása volt, egyszerű fizika hozzárendelése a modellhez. Adtam egy Rigidbody komponenst a modellnek, amely tulajdonképpen lehetővé teszi, hogy egy adott testre hatással legyenek fizikai erők valamint a főbb tagjainak Mesh Collidereket, hogy tudjon ütközni különböző elemekkel és a tereppel. [8] [9] Ezekután létrehoztam egy egyszerű scriptet MovementController.cs néven, amely primitív módon képes volt vezérelni a járművet. Ennek a scriptnek a fontos részei alapvetően azok a kódrészletek voltak, amelyek a végső verzióban is megmaradtak, ezek a kamera vezérlése, a célzás megvalósítása harmadik személy nézetből (kis utólagos kiegészítéssel) és a felhasználói inputok kezelése.

3.2 Felfüggesztés és lánctalp

A modell nem rendelkezett semmiféle mozgásra alkalmas elemmel, így a felfüggesztést és a lánctalpat is teljes mértékben saját kézzel kellett legyártanom.

3.2.1 Felfüggesztés

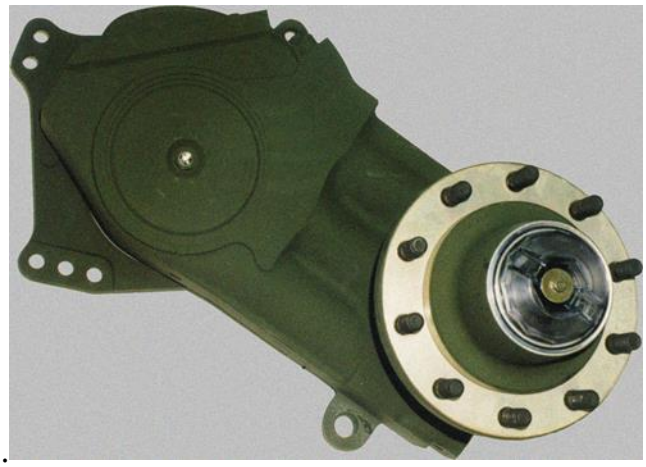
Alapvetően a lánctalp 7-7 alsó görgőjéhez kellett felfüggesztést csatolnom, a többi görgő pozíciójukból adódóan nem igényelt rugózást. Ennek megvalósításához több különböző megoldás közül kellett kiválasztanom a legmegfelelőbbet. Végül három lehetőségre szűkítettem őket:

- Wheel Collider
- Spring Joint
- Hinge Joint

A Wheel Collider alapvetően egy sok az egyben megoldásnak tűnt, hiszen mindamellett, hogy lehetőséget biztosít a felfüggesztés testreszabására, rögtön alkalmas egy collider hozzárendelésére is, valamint a kerekek forgatását is meg tudja valósítani. [10] Viszont a működésének elve nem igazán hasonlít az általam kreálni kívánt hidropneumatikus felfüggesztéséhez.

A Spring Joint a nevéből adódóan elsőre az egyértelmű logikus választásnak tűnt, de rövid vizsgálat után világossá vált, hogy alapvetően egyszerű rugók megvalósításához hasznos, amelyek oldalra is ki tudnak lengeni, amely nem felelt meg a modell elvárásainak. [11]

A Hinge Joint, a nevéből könnyen következtethető módon főleg csuklók megvalósítására lett kitalálva, ilyenek például az ajtók és az ingák, de szerencsére lehetőséget biztosít egy változtatható rugóerő megadására is amely egy adott irányba húzza azt. Ennek segítségével lehetséges volt a valódihoz nagyon hasonló elven működő felfüggesztést létrehozni. [12]

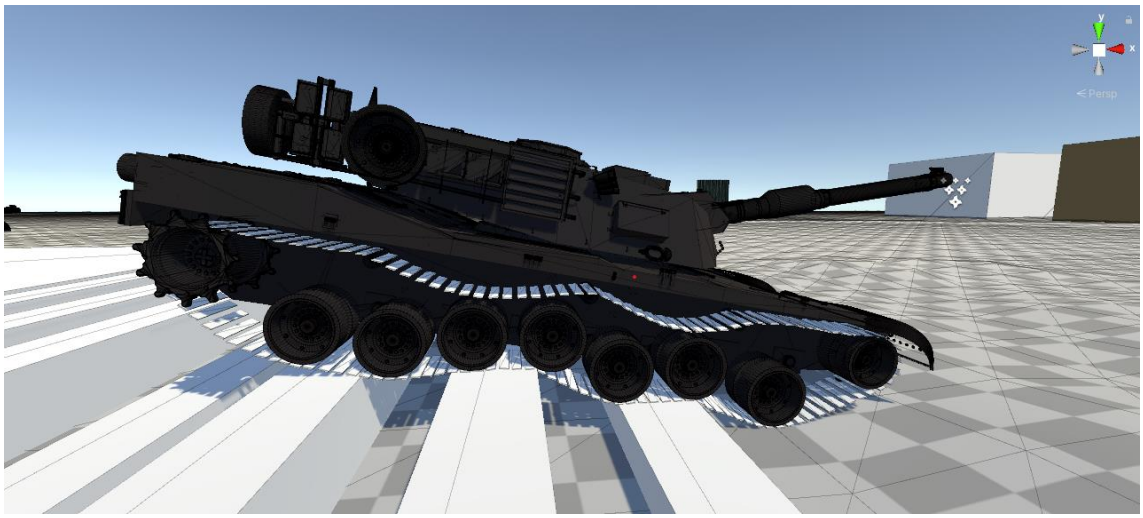


3.2.1 ábra: Hidropneumatikus felfüggesztés

A hidropneumatikus felfüggesztést (3.2.1. ábra) így végül a Unity Hinge Joint komponensének segítségével valósítottam meg. Minden egyes felfüggesztésdarab összességében 2 ilyen elemből állt. Az egyik a rugózásért volt felelős, a másik pedig a görgők szabad forgását tette lehetővé. Előbbi tehát rendelkezett egy, a jármű tömegéhez arányosan megválasztott rugóerővel, hogy alapesetben a test ne üljön túl mélyen, viszont alkalmas legyen a terepakadályokon reálisan le és fel mozogni. Ezek mellett fontos volt beállítani a megfelelő tengelyeket is, hogy az adott felfüggesztés mely pont körül forduljon el. Szerencsére a Unity beépítetten rendelkezik ilyen paraméterekkel. A másik Hinge Joint pedig ehhez lett csatlakoztatva és a kerekek szabad forgását tette lehetővé.

3.2.1.1 A felfüggesztés működése különböző terepeken

Több terepviszonyt is létrehoztam a projektben, hogy a felfüggesztés működését minél szélesebb körben tudjam tesztelni. A felfüggesztés működését a 3.2.1.1-1. és 3.2.1.1-2. ábra mutatja be.



3.2.1.1-1. ábra: Felfüggesztés tesztelése

3.2.1.1.1-2. ábra: Felfüggesztés tesztelése

Szemmel látható módon sikeresen felvette a terep adottságait minden görgő, ahogy ez a két ábrán is látható.

3.2.1.2 A felfüggesztés kézi állítása

A hidropneumatikus felfüggesztés állíthatóságát a már korábban említett MovementController.cs-ben valósítottam meg: paraméterként megkapja a rugózó felfüggesztéseket, majd ezek erősségét felhasználói inputtól függően állítja be, két határérték között, adott lépésekkel:

```
        //Hydropneumatic susepension behaviour
        //Move down
        if   (Input.GetKey(KeyCode.R)   &&   !Input.GetKey(KeyCode.F)   &&
currentSpringForce - (suspensionStep*3) > minSpringForce)
        {
            currentSpringForce -= (suspensionStep*3);
            int i = -3;
            foreach (var hinge in leftSuspension)
            {
                JointSpring spring = hinge.spring;
                spring.spring += suspensionStep * i;
                hinge.spring = spring;
                i++;
            }

            i = -3;
            foreach (var hinge in rightSuspension)
            {
                JointSpring spring = hinge.spring;
                spring.spring += suspensionStep * i;
                hinge.spring = spring;
                i++;
            }
        }

        //Move up
        if   (Input.GetKey(KeyCode.F)   &&   !Input.GetKey(KeyCode.R)   &&
currentSpringForce + (suspensionStep * 3) < maxSpringForce)
        {
            currentSpringForce += (suspensionStep * 3);
            int i = -3;
            foreach (var hinge in leftSuspension)
            {
                JointSpring spring = hinge.spring;
                spring.spring -= suspensionStep * i;
                hinge.spring = spring;
                i++;
            }

            i = -3;
            foreach (var hinge in rightSuspension)
            {
                JointSpring spring = hinge.spring;
                spring.spring -= suspensionStep * i;
                hinge.spring = spring;
                i++;
            }
        }
```

A végeredmény a 3.2.1.2 ábrán látható.

3.2.1.2. ábra: Hidropneumatikus felfüggesztés tesztelése

3.2.1.3 A felfüggesztés hatása a célzásra

A hidropneumatikus felfüggesztés az egész jármű testét döntötte előre és hátra, amely a célzásra is kihatással volt, ezért kompenzálni kellett az ágyú vertikális forgását a test szögétől függően, amelyet a torony szöge is befolyásol, hiszen ha a torony oldalra néz derékszögben a testhez képest, akkor nem kell kompenzálni, ha pedig egyirányba néznek, akkor a test teljes vízszintessel bezárt szögével kell ezt megtenni.

Mielőtt ennek pontos megvalósítását taglalnám, fontosnak tartom, hogy kitérjek a célzás működésére is. Alapvetően a demonstrációs projekt egy harmadik személy szögből történő célzást valósít meg. [13] A tornyot az egér vízszintes mozgatásával lehet forgatni. A függőleges célzás viszont egy fokkal komplikáltabb. A kamera pozíciójából kilövünk egy sugarat abba az irányba, amelybe a kamera néz. Amennyiben ez a sugár ütközik valamilyen felülettel (talaj, valamilyen objektum), akkor ezt az ütközési pontot vesszük. Amennyiben nem, akkor megfelelően nagy távolságban veszünk egy pontot amelyen ez a sugár áthalad:

```
RaycastHit hit;
Ray ray = new Ray(cam.position, cam.TransformDirection(Vector3.forward));

// Does the ray intersect any objects
if (Physics.Raycast(ray, out hit, Mathf.Infinity, layerMask))
{
     //gets the hit position
}
else
{
     //set position if no hit with raycast
     hit.point = ray.origin + ray.direction * 1000;
}
```

A célzott pont kiválasztása után az ágyút megfelelő szögbe kell állítani ahhoz, hogy abba az irányba nézzen, ahova lőni szeretnénk. Ennek megállapításához két kétdimenziós vektor szögét vesszük, amelybe a löveget kell forgatnunk. Az egyik vektor az ágyúból a torony forgásával megegyező irányban mutat vízszintesen előre, a másik pedig az löveget köti össze a célzott ponttal. Persze ez a szög mindig pozitív értéket fog adni, ezért meg kell néznünk, hogy a célzott pont függőlegesen feljebb vagy lejjebb van-e az ágyúhoz képest, és ennek függvényében kell beállítanunk a szög előjelét.

Ezek mellett szükség van kompenzálni ezt a szöget a test hosszanti elfordulásával is, melyet a felfüggesztés okoz. Ez a kompenzációs szög a torony elfordulásának a koszinuszának és a test hosszanti irányú vízszintessel bezárt szögének a szorzata. A Unity a legtöbb grafikus motorhoz hasonlóan kvaterniókban tárolja az objektumok forgását, de az eulerAngles tulajdonság lehetővé teszi, hogy ezeket könnyebben kezelhető szögekké konvertáljuk. [14]

```
     Vector2 gunRelativePos = new Vector2(Mathf.Sqrt((hit.point.x -
gunAxis.position.x) * (hit.point.x - gunAxis.position.x) + (hit.point.z -
gunAxis.position.z) * (hit.point.z - gunAxis.position.z)), hit.point.y -
gunAxis.position.y);

     float gunAngle = Vector2.Angle(gunRelativePos, new
Vector2(Mathf.Sqrt((hit.point.x - gunAxis.position.x) * (hit.point.x -
gunAxis.position.x) + (hit.point.z - gunAxis.position.z) * (hit.point.z -
gunAxis.position.z)), 0f));
     if (hit.point.y < gunAxis.position.y)
     {
          gunAngle = -gunAngle;
     }
     //Compensate for hull rotation
     float hullRotation = transform.localEulerAngles.z;
     if (hullRotation > 180)
     {
          hullRotation = hullRotation - 360;
     }
          gunAngle -= Mathf.Cos(turret.localEulerAngles.y *
Mathf.Deg2Rad) * hullRotation;
```

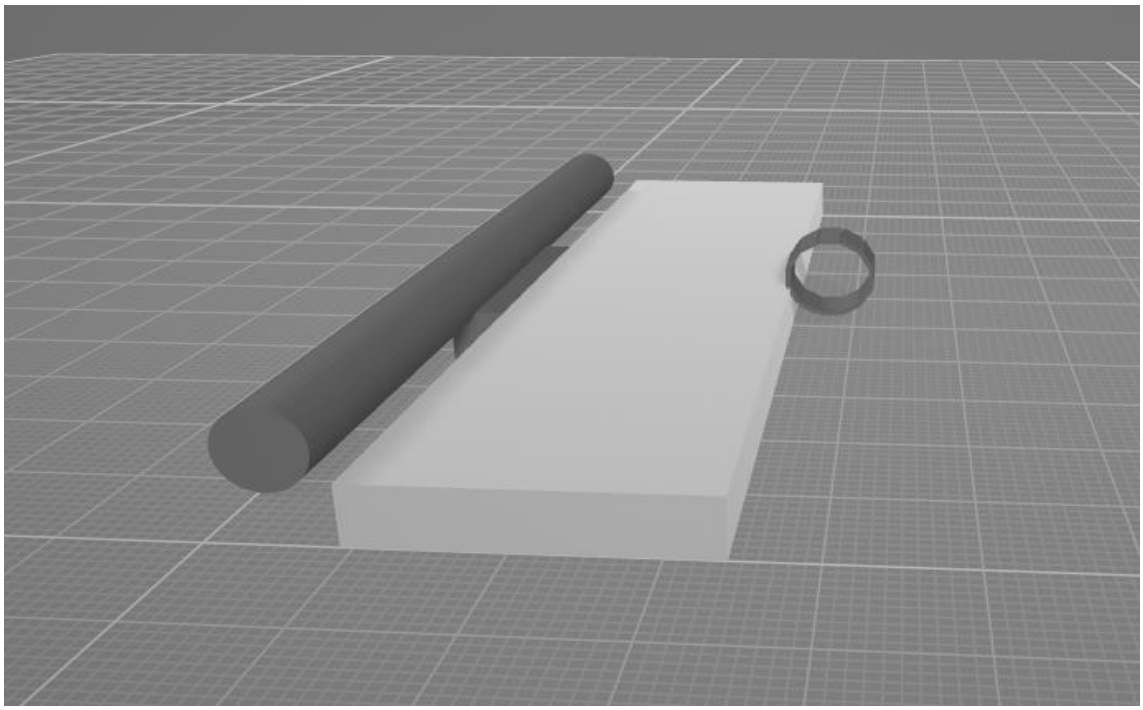
Végül pedig ezt még annyival kell kiegészíteni, hogy minden tank ágyúját csak bizonyos mértékben lehet függőlegesen kitéríteni, így amennyiben a kiszámolt szög kívül esne az két szélsőérték közti intervallumon, akkor mindaddig nem kell forgatni a löveget, ameddig nem kerülünk újra az intervallumon belülre. Ez azt okozza, hogy a lövés nem mindig abba az irányba repül, amerre a célkereszt mutat.

3.2.2 A lánctalp

A lánctalpak kialakításához három lehetséges megoldás tűnt kivitelezhetőnek:

- A lánctalpak valódi háromdimenziós modellezése, a köztük lévő kapcsolat valós kialakításával
- A lánctalpak modellezése valamilyen modellezőszoftver segítségével (pl. Blender), melyben az adott szoftver lehetőségeit kihasználva alakítjuk ki a kapcsolatot az elemek között
- A lánctalpak egyszerűbb modellezése téglatestekkel és köztük a Unity által támogatott Joint-ok megfelelő definiálása, a valós viselkedés szimuláláshoz

Annak ellenére, hogy az első lehetőség alapvetően valódibb modellezést tehetne lehetővé, több komoly negatívummal is rendelkezik. Az egyik komoly probléma az, hogy a PhysX megkötései miatt nem lehet Rigidbody-khoz konkáv Collider-eket rendelni, de ezt a problémát ki lehet küszöbölni azzal, hogy a konkáv Collidert több konvex társával helyettesítjük (Compound Collider). [15] Létre is hoztam egy ilyen lánctalpat Blenderben (3.2.2. ábra), de sajnos ennyi folyamatos ütközés feldolgozása nem bizonyult működőképes megoldásnak, az alacsony FPS mellett gyakran az ütközések feldolgozása is pontatlan volt.



3.2.2. ábra: Compond Collider-rel rendelkező lánctalpdarab

A projekt alapvetően a Unity által kínált komponensek használatára fókuszált, nem pedig a háromdimenziós szoftverek használatára, ezért habár jó megoldásnak tűnt a második opció (Blenderben Bendy Bone-ok használata), inkább a harmadik mellett döntöttem.

3.2.2.1 A lánctalpak modellezése

A lánctalpdarabokat téglatestekkel modelleztem, ezekhez Box Collidereket rendelve, ezzel optimalizálva a rendszert, hiszen jelentősen gyorsabb ezek fizikai szimulációja, komplikáltabb Mesh Colliderekhez képest. Persze vizuálisan ettől függetlenül lehetett volna a téglatesteknél komplikáltabb formát alkalmazni, de ez a változtatás könnyen implementálható a jelenlegi projektbe, viszont a szakdolgozat célja alapvetően nem a háromdimenziós design.

3.2.2.2 A lánctalpak összekapcsolása

A lánctalpak összekapcsolásához a Unity által kínált Joint-okat használtam. Ezek széleskörű módon képesek egymáshoz csatlakozni. [16] Számomra két fontos kapcsolat létrehozása volt fontos:

- Képesek legyenek elforogni egymás mellett, egy adott tengely körül
- Ne hullámozzanak túlságosan nagy mértékben, feszüljenek meg

Előbbire nyújt kitűnő megoldást a már korábban is használt Hinge Joint. Kézzel megadott forgástengelyekkel kifejezetten jó eredményt kapunk. A 3.2.2.2. ábrán látható példával megegyező módon sorban, egyesével összekötöttem a megfelelő lánctalpdarabokat, ehhez a Connected Body paramétert állítottam be a következő elemre. Az Anchor és Axis paraméterek segítségével beállítottam a forgástengelyt. Továbbá kikapcsoltam az előfeldolgozást is, mert az esetek jelentős százalékában hibás szimulációt generált.

Az elemek Rigidbody-jának beállítottam, hogy csak az elvárt irányban tudjanak mozogni és forogni, ezzel is optimalizálva a program futását, hiszen nem kellett olyan síneket, határokat kialakítani, amelyek limitálják a láncelemek pozícióját, további fizikai számításokat generálva.

Az így kialakult kinézet azonban távolról sem felelt meg, mert a lánctalp nem volt megfelelően feszes, valamint néhány esetben túlságosan nagy mértékben hullámzott, vagy távolodtak el az elemek egymástól. Ezeket a hibákat elkerülendő,

limitáltam a lánctalpak egymással bezárt szögének minimumát és maximumát, amely a valós esetben is létezik, valamint tettem egy gyenge rugót a Hinge Joint-ra, hogy a lánctalpak igyekezzenek egymással párhuzamosan elhelyezkedni egymás mellett, ahogy ez a valóságban is történik. Utóbbi rúgó beállításával a nagymértékű hullámzást is képes voltam csökkenteni, ezt a Damper paraméter értékének változtatásával lehetett megoldani. A Damper tulajdonképpen azt adja meg, hogy a rugó hullámzása milyen mértékben csillapodjon, a lengéscsillapításért felelős. A beállítások a 3.2.2.2 ábrán vizuálisan is megtekinthetőek.

3.2.2.2. ábra

Ezen beállítások után jelentősen jobbnak tűnt a szimuláció, viszont továbbra sem volt elég feszes a lánctalp. Elsőre Spring Joint-ok segítségével próbáltam csökkenteni a láncelemek távolságát, de sajnos ez nem vezetett sikerre, ezért más módon kellett megoldani a problémát. Végül a legjobb eredményt adó megoldás a lánctalpak kifeszítésére a teljes lánctalp méretének csökkentése volt, így amikor elindul a szimuláció, meg kell nyúlniuk ahhoz, hogy körbe érjenek a görgők körül.

3.2.2.3 A lánctalp meghajtása

A lánctalpakat mindkét oldalon a hátulsó görgő hajtja meg, a többi görgő nem rendelkezik motorikus tulajdonságokkal. Szerencsére a Hinge Joint ebben az esetben is kapóra jön, hiszen lehetőséget nyújt Motor hozzárendeléséhez is. Ennek paramétereként megadhatjuk a motor erejét, a célsebességet, valamint, hogy a célsebesség túllépése esetén fékként működjön-e. Ezeket a paramétereket a TrackRotator.cs script adja meg a motornak, amely pedig a célsebességet a MovementController.cs-ből kapja, amely pedig a felhasználói inputokból állapítja meg azt.

Ezzel elértük, hogy felhasználói inputra megfelelő módon forgassuk a hátsó görgőket, viszont az egész lánctalpat meg kell hajtani valamilyen módon. Ennek megvalósítására két kézenfekvő opció áll rendelkezésünkre:

- Fizikailag úgy kialakítani a hátsó görgőt, hogy beleakadjanak a lánctalpelemek, így meghajtva a teljes lánctalpat
- A Unity által biztosított Fixed Joint segítségével összetapasztani a lánctalpelemeket a hátsó görgővel

Ahogyan eddig is, most is a Unity által biztosított Joint-okat használtam a megoldáshoz, de ettől függetlenül úgy gondolom, hogy ebben az esetben a görgő megfelelő háromdimenziós modellezése, valamint annak Compound Colliderének kialakítása is pontos és gyors megoldást biztosíthat.

A Fixed Joint lehetőséget ad két elem olyan módú összekapcsolására, mintha szülő-gyerek viszonyban lennének az editorban. Tehát a csatlakoztatott elem együtt fog mozogni a másikkal. A TrackRotator.cs script ezt a viselkedést valósítja meg dinamikus módon. Ez azt jelenti, hogy mindig az aktuálisan meghajtott elemhez köti a Fixed Joint-ot. Az aktuálisan meghajtott elem meghatározása nem túl nehéz, azt az elemet kell kiválasztani, amely pozícióját tekintve a görgő mögött van, és ezen elemek közül a legkisebb a görgőhöz képesti távolságának a függőleges vetülete.

3.2.2.4 Egyéb beállítások a lánctalp működéséhez

Mivel a lánctalp pontos szimulációjához az alapértelmezett Solver iterációk száma nem volt elegendő, ezért ennek értékét 6-ról 10-re növeltem, valamint a Unity Fixed Timestep-jét 0,01-re állítottam, amely azt írja le, hogy milyen gyakran futnak le a

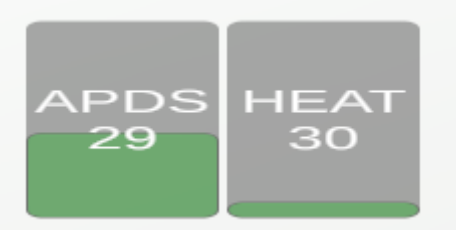
FixedUpdate függvények a játék objektumain. [17] Ezen beállítások alkalmazása után is 300 FPS körüli értékeket kaptam a kibuildelt verzióban.

3.3 Lövedéktípusok

A projekt keretein belül kétféle lövedéktípust valósítottam meg, az egyik egy kinetikus energiával működő lövedék, a másik pedig egy HEAT (High-Explosive Anti-Tank) lövedék.

3.3.1 UI, lövedékek közti váltás és újratöltés

Létrehoztam egy egyszerű UI-t, mellyel követhetjük a jelenleg rendelkezésünkre álló lövedékek számát, típustól függően, valamint indikálja a lövedékek újratöltését, és azt, hogy jelenleg éppen milyen lövedéket töltünk a lövegbe vagy milyen van már betöltve. Ennek a kialakítását a 3.3.1. ábrán láthatjuk.



3.3.1. ábra: UI a lövedékekhez

A MovementController.cs scriptben beállíthatjuk a löveg újratöltési idejét, valamint itt tárolódnak el a rendelkezésünkre álló lövedékek is, de itt található az a kódrészlet is, amely a felhasználói inputokat dolgozza fel. Az ágyú csak akkor tud tüzelni, ha van benne lövedék, amit egy bool változó jelez. Minden lövés vagy lövedéktípus csere után ez a bool hamis lesz, és egy Invoke segítségével pontosan a megadott újratöltési idő után lesz ismét igaz, ha addig nem változtatjuk meg újra a kilőni kívánt lövedék típusát. A lövedéktípusok között az 1-es és 2-es gombokkal lehet váltani, az egyessel az APDS lövedéket, a 2-essel a HEAT lövedéket tudjuk kiválasztani.

Amikor ez az újratöltés elindul, akkor a UI-on is elkezd a megfelelő Slider felfelé csúszni, amely azt indikálja, hogy a lövedék jelenleg hol tart a betöltési időben. Amennyiben a csúszka teljesen felér, azután tudunk lőni.

Ha valamelyik lövedéktípusból kifogy a lőszerrekesz, akkor automatikusan a következő lövedéktípus kezd el betöltődni. Olyan típusra nem tudunk váltani, amellyel már nem rendelkezünk. Ha nem rendelkezünk lövedékkel, akkor lőni sem tudunk.

A lövedékek betöltési ideje nem függ a lövedék típusától, ez az érték a járműtől függ.

3.3.2 Találatok detektálása, effektív páncélvastagság kiszámítása

A találatok detektálását a Unity OnCollisionEnter függvénye végzi, amelyben megkapjuk paraméterként azt a páncéllemezt, egyéb tárgyat, amellyel a lövedék ütközött. [18]

Fontos hozzátenni, hogy amikor ez a függvény meghívásra kerül, akkor a lövedék már nem azzal a sebességvektorral rendelkezik, mint a becsapódás előtti pillanatban, ezért ezt mindig el kell mentenünk az előző FixedUpdate-ben, hogy helyesen meg tudjuk határozni a becsapódás végkimenetelét.

A normalizáció előtti effektív páncélvastagság minden lövedék esetében ugyanakkora. Ennek kiszámításához elég megállapítanunk a páncéllemez normálvektora és a lövedék becsapódás előtti irányvektora által bezárt szöget (beesési szöget), valamint az adott páncéllemez vastagságát.

Minden páncéllal ellátott objektum rendelkezik egy Deformable scripttel, amely alapvetően a páncél deformációjáért felelős, de itt vannak eltárolva a tárgy páncéllemezeinek vastagságai is. Egy Deformable objektum modelljének minden háromszöge egyedi páncélvastagsági értékkel rendelkezhet, de optimalizáció gyanánt lehetőség van arra, hogy egy egész Deformable-t homogén páncélvastagsággal jellemezzünk, mindemellett viszont gyakorlati haszna is van, hiszen a valóságban is léteznek homogén vastagságú tárgyak.

Az eltalált háromszög meghatározásához végigiterálunk az eltalált modell összes háromszögén, és megkeressük azt, amelynek tömegközéppontja a legközelebb helyezkedik a becsapódás pontjához. Ez egy kifejezetten erőforrásigényes feladat, ezért érdemes minél több helyen a homogén páncélvastagságot preferálni, ahol ez a lépés kimarad.

Kiegészítésképpen hozzáfűzném, hogy azért nem használhatjuk például az ennél a módszernél sokkal költségeffektívebb Raycast általi megoldást, ahol egy megfelelő irányú sugár kilövésével meghatározhatjuk iterációk nélkül, hogy mely háromszöget találtuk el, mert a későbbi részekben dokumentált páncéldeformációk konkáv alakokká alakítják a Deformable objektumokat, viszont a Unity csak konvex colliderekkel

rendelkező Rigidbody-k létrehozását engedélyezi. Ezért a collidereknek vagy konvexeknek kell lennie – mint jelen esetben -, vagy a deformációk során a modelleket futási időben kell konvex colliderekre felbontani, amely egy nagyon költséges művelet, tehát igazából nincs egyértelműen ideális választás. Ha a colliderek konvexek, akkor viszont a grafikusan megjelenített mesh és a collider nem ugyanazokból a háromszögekből fog állni, így hibás értékeket kapnánk.

Miután megkaptuk a beesési szöget és az eltalált háromszög vastagságát, egy egyszerű képletbe behelyettesítve megkapjuk az effektív páncélvastagság értékét:

effektív vastagság = páncéllemez vastagsága / cos (beesési szög)

3.3.3 Lövedéktípusok

Mindkét lövedéktípus külön-külön scripttel rendelkezik, amelyek eltárolják azok fontos paramétereit, detektálják az ütközéseket, valamint kiszámolják, hogy a lövedék gellert kap, átüti a páncélt, vagy esetleg elakad a páncéllemezben találat esetén. Azt, hogy a lövedékek becsapódásának mi a végkimenetele, egy rövid üzenet jeleníti meg.

3.3.3.1 Kinematikus energiával működő lövedékek

Ezeknek a lövedékeknek nagyobb a torkolati sebessége, mint a HEAT lövedékeknek, viszont cserébe az átütési értékük alacsonyabb, amely a megtett távolsággal arányosan tovább csökken.

Az átütőérték csökkenését az alábbi módon kapjuk meg:

```
distanceCovered = rb.velocity.magnitude * Time.fixedDeltaTime;
penetration -= Mathf.RoundToInt(distanceCovered * penetrationLossOverDistance);
```

ahol, a distanceCovered az előző vizsgált ponthoz képest megtett távolság, a penetrationLossOverDistance pedig egy arányszám, amellyel beállíthatjuk, hogy mekkora legyen az átütés értékének csökkenése.

Ezek a lövedékek rendelkeznek normalizációval is, tehát amikor egy páncéllemezzel találkoznak, akkor egy adott mértékben csökkentik a beesési szöget, ezzel csökkentve az effektív páncélvastagságot is. Jelen modellben ezt egy adott szöggel teszik, természetesen a beesési szög értéke nem lehet negatív.

A másik képességük, hogy nem robbannak fel rögtön találat esetén, így például ha gellert kapnak, akkor tovább folytatják útjukat, persze az átütőerejükből sokat vesztve.

Ennek megvalósítása kódban:

```
if (oldAngle > 75)
{
     penetration -= 50;
     penetrationMessage.SetMessage("Ricochet!");
}
else if (penetration >= armourValue / Mathf.Cos(angle))
{
     deformable.Collide(velocityBeforePhysicsUpdate, rb.position);

     penetrationMessage.SetMessage("Penetration!");
     Destroy(gameObject);
}
else
{
     deformable.Collide(velocityBeforePhysicsUpdate, rb.position);

     penetrationMessage.SetMessage("No penetration!");
     Destroy(gameObject);
}
```

A penetrationMessage SetMessage függvénye segítségével megjelenítjük a képernyőn, hogy a kilőtt lövedék végül átütötte-e a páncélt. Ez a felirat a 3.3.3.1. ábrán látható módon néz ki. A felirat 3 másodpercig látszódik.

3.3.3.1. ábra: Gellert jelző felirat

3.3.3.2 HEAT lövedékek

A High-Explosive Anti-Tank lövedékek működésének megvalósítása sok szempontból hasonlít a kinematikusakéhoz.

Ezeknek a lövedékeknek viszont az átütőereje nem csökken a megtett távolság függvényében, nem hat rájuk normalizáció, ezért modellezésük sok szempontból egyszerűbb. Az egyetlen különlegességük, hogy amikor gellert kapnak, akkor nem vesztenek átütési erejükből. Ugyanakkor ez nem gyakran fordul elő, mert ahhoz, hogy gellert kapjanak, rendkívül nagy szögben kell becsapódniuk.

3.3.4 Lövedékek kilövése, visszarúgás

A lövedékek kilövését az egér bal gombjának kattintásával kezdeményezhetjük. Ezt, ahogyan minden más felhasználói inputot is, a MovementController.cs script dolgoz fel.

3.3.4.1 Lövedék fizikai megvalósítása

Minden lövedék rendelkezik egy fizikai modellel, amely egy Mesh Collider és egy Rigidbody komponensből áll, valamint rendelkezik egy, már a korábbiakban részletezett scripttel is. Lövéskor az ágyú csövében létrehozunk egy ilyen lövedékpéldányt a Unity Instantiate metódusával, melyben meghatározzuk a lövedék pozícióját és elforgatását úgy, hogy az a megfelelő helyen és irányban legyen. Mivel a modellek, amelyek alapján a lövedékeket létrehozzuk inaktív objektumok, ezért a létrehozott lövedékeket aktívvá kell tenni, majd a Rigidbody-jának sebességét a lövedéktípustól függően kell beállítanunk, persze ennek iránya is azonos az ágyú állásának irányával.

```
GameObject currentAmmo = Instantiate(apfsdsAmmo,
ammoPosition.position, turret.rotation * Quaternion.Euler(0, 90, 0) *
Quaternion.Euler(-currentGunRotation, 0, 0));
currentAmmo.SetActive(true);
Object.Destroy(currentAmmo, 5f);
currentAmmo.GetComponent<Rigidbody>().velocity = gunAxis.right *
ApfsdsScript.startVelocity;
```

Annak érdekében, hogy feleslegesen ne pazaroljunk memóriát a kilőtt lövedékek tárolására, minden lövedék legfeljebb 5 másodperc után megsemmisül.

3.3.4.2 Visszarúgás

Newton III. törvénye értelmében, ha kilövünk egy lövedéket a lövegből, akkor a lövegre is hat egy, a lövedék haladási irányával ellentétes irányú erő. [19] A projektben ezt az erőt paraméterként adhatjuk meg, irányát pedig az alábbi módon kaphatjuk meg:

```
var horizontalRecoil = Mathf.Cos(verticalRotation);
var yRecoil = -Mathf.Sin(verticalRotation);
var xRecoil = -Mathf.Cos(horizontalRotation) * horizontalRecoil;
var zRecoil = -Mathf.Sin(horizontalRotation) * horizontalRecoil;
```

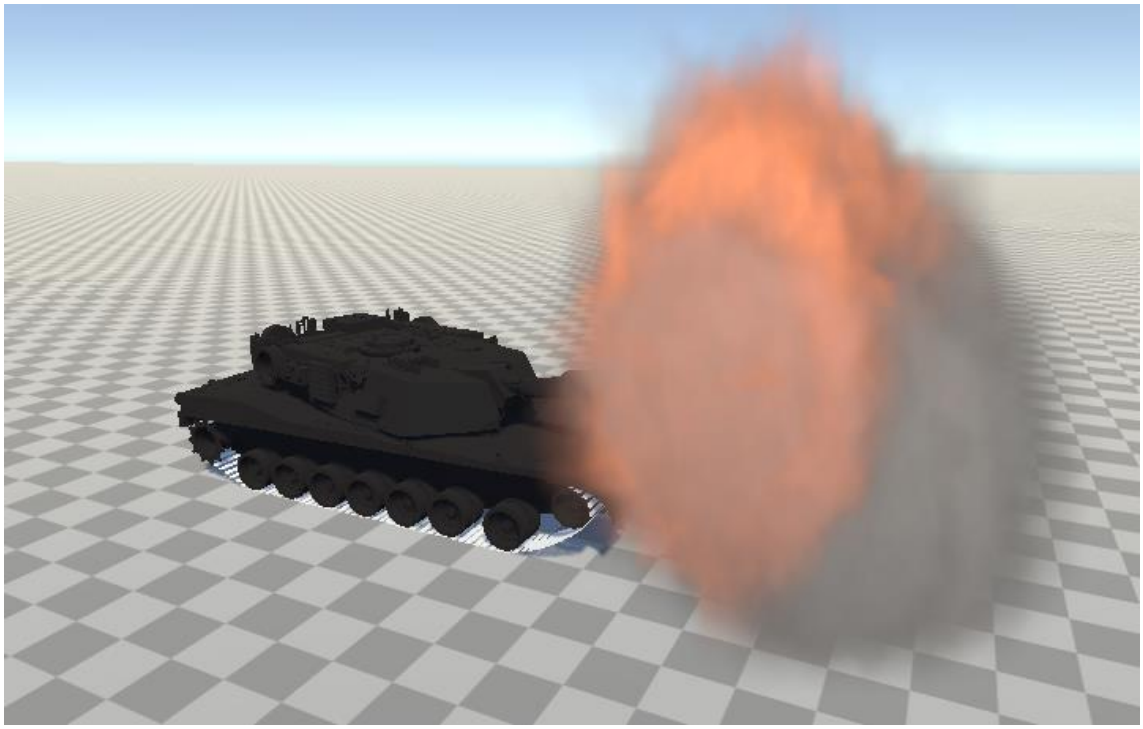
A horizontalRotation a torony szöge a jármű testéhez képest, a verticalRotation pedig az ágyú vízszintessel bezárt szöge. Az xRecoil, yRecoil és zRecoil a visszarúgás irányának x, y és z komponensei.

Ez az erő a jármű testének Rigidbody-jára hat, a löveg pozíciójánál. Ezt a Unity motorban a RigidBody beépített AddForceAtPosition függvényével valósíthatjuk meg.

```
rb.AddForceAtPosition(new Vector3(explosionForce * xRecoil, explosionForce
* yRecoil, explosionForce * zRecoil), gunAxis.transform.position);
```

3.3.4.3 Lövés effekt

A lövedékek kilövéséhez létrehoztam egy effektet is, amely valósághűbbé teszi a robbanás pillanatát. Ezt a Unity Particle System komponensével készítettem el. [20] A Unity Asset Store-on található egy Unity Particle Pack nevű bővítmény, melyet a játékmotor gyártója kreált. [21] Ebben található néhány Prefab (előre létrehozott effekt), melyek közül kiválasztottam azt, amelyik a legjobban hasonlított egy ágyúlövésre, majd a megfelelő módosítások után hozzárendeltem a lövésekhez. Végeredményképpen az effekt a 3.3.4.3. ábrán látható módon nézett ki.



3.3.4.3. ábra: A lövés effekt

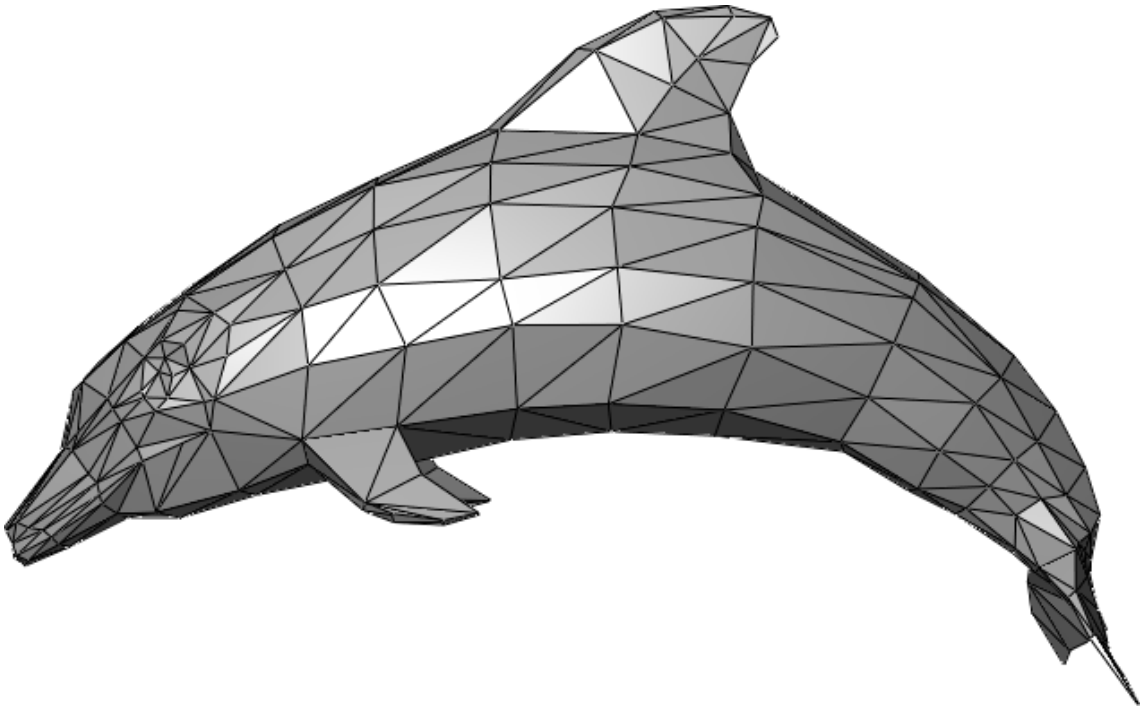
3.4 A löveg által kilőtt lövedékek hatása a páncéllemezek ellen

A Deformable scripttel rendelkező elemek a pályán őket érő találat esetén megfelelő módon változtatják alakjukat, horpadnak be.

3.4.1 A Unity modellek háromszöghálója

A számítógépes háromdimenziós modellek úgynevezett polygon mesh-ből állnak, amely azt jelenti, hogy a modell teljes felépítése egy sokszögháló. [22] Erre mutat példát a 3.4.1. ábra. Minél több sokszögből áll egy modell, annál inkább lehetséges részletessé tenni.

A Unity esetében ezek a polygon mesh-ek igazából csak háromszögekből állnak. Ez lehetséges, hiszen minden sokszöget fel lehet bontani háromszögekre úgy, hogy a háromszögek csúcsai megegyezzenek a felosztani kívánt sokszög csúcsaival.



3.4.1. ábra: Példa háromszöghálóra

Minden Deformable scripttel rendelkező Unity modell rendelkezik egy MeshFilter komponenssel, amelynek van egy mesh tulajdonsága. Ez a mesh tulajdonság tárolja el a háromszöghálót, amely a következőképpen épül fel:

- A háló csúcsai egy Vector3 tömbben tárolódnak el, amely a mesh vertices tulajdonságában található
- Az, hogy melyik háromszög mely csúcsokból áll, azt a mesh triangles tulajdonsága határozza meg. Ez egy int tömb, melynek minden három eleme leír egy háromszöget, az egyes elemek értéke pedig azt mutatja meg, hogy az adott csúcs mely indexű a csúcsokat eltároló tömbben. Például, ha tömb 0., 1. és 2. elemének értéke 4, 6 és 9 akkor a haló egyik háromszögének csúcsai a vertices tömb 4., 6. és 9. eleme. [23]

3.4.2 Páncéllemezek felbontása

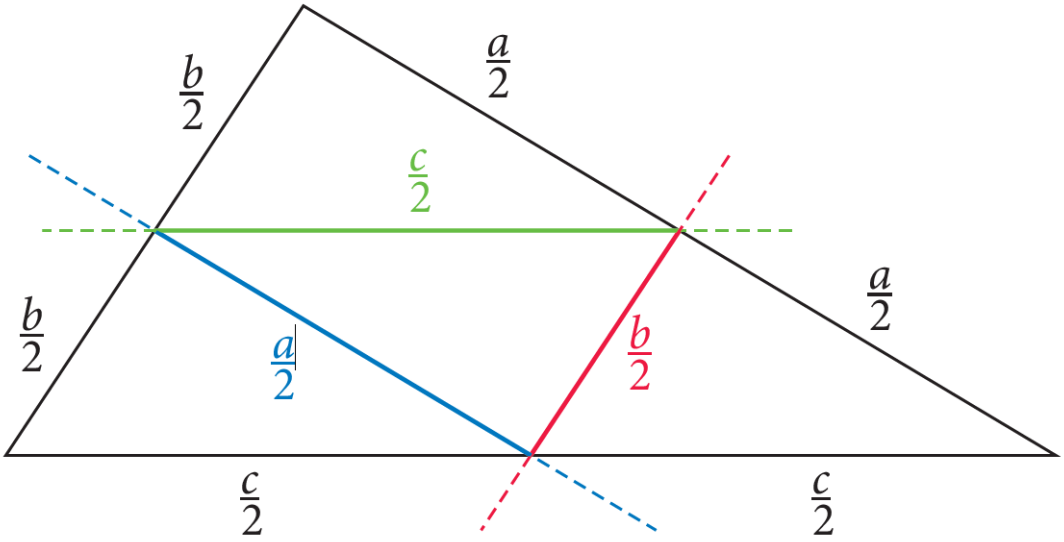
Ahhoz, hogy a becsapódás utáni horpadás valóságszerűen nézzen ki, szükséges, hogy az objektum háromszöghálója megfelelően sűrű legyen. Például, ha egy nagy téglalapot találunk el, amelynek háromszöghálója mindössze két háromszögből áll, akkor csupán ezek csúcsainak mozgatásával nem lehetséges a deformáció élethű szimulációja. Szerencsére a Unity lehetőséget nyújt arra, hogy az objektumok háromszöghálóját futásidőben módosítsuk, melyet a korábban említett mesh tulajdonság változtatásával érhetünk el. Alapvetően kétféle felbontási módot hoztam létre, az egyik a háromszög súlyvonalai, a másik pedig a háromszög középvonalai mentén bontja fel a hálót. Minden Deformable elemre külön-külön definiálhatjuk, hogy melyik felbontási mód hányszor fusson le, hogy megfelelő eredményt kapjunk, de két limit is meg van határozva:

- Területi: amelynél kisebb területű háromszögek nem kerülnek feldarabolásra [24]
- Távolsági:
  - Középvonalas feldarabolás esetén: amelynél távolabb található háromszögek nem kerülnek feldarabolásra
  - Súlypontos feldarabolás esetén: amelynél közelebbi háromszögek mindenképpen feldarabolásra kerülnek

Az újonnan létrehozott háromszögek páncélvastagsága megegyezik a kiinduló háromszögével.

3.4.2.1 Felbontás a középvonalak mentén

A háromszög középvonalai a háromszög oldalainak felezőpontjait kötik össze, a 3.4.2.1. ábrán látható módon. [25]



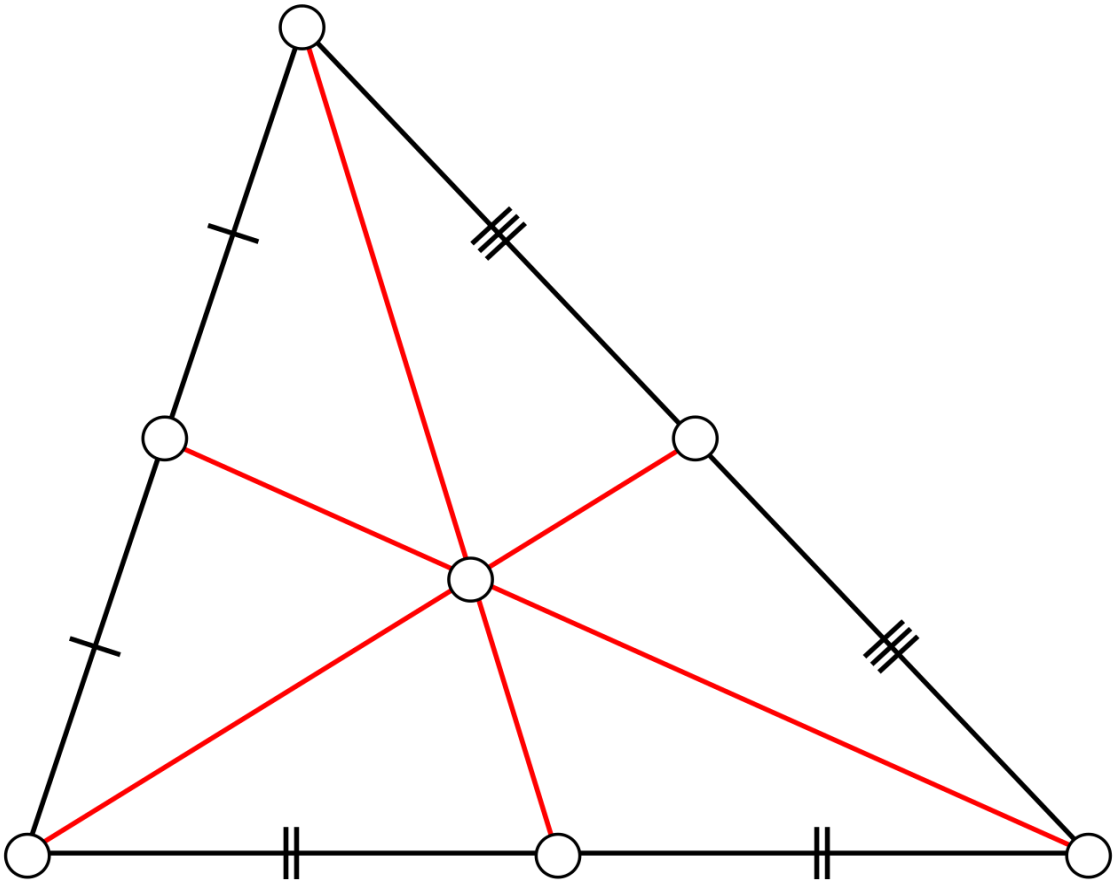
3.4.2.1. ábra: Háromszög középvonalai

Ezek a vonalak a háromszöget 4, az eredeti háromszöghöz hasonló, kisebb háromszögre bontják, melyek egyforma területűek, így a háló felbontásának homogenitása nem változik. [26]

Továbbá, ahogyan már korábban említettem, paraméterként meg lehet adni, hogy a találat mely sugarú körében található háromszögek kerüljenek felbontásra (a találat és a háromszög súlypontjának távolsága a megadott értéknél kisebb legyen), így optimalizálva a felbontást, mert egy adott távolágon túl a deformáció mértéke annyira alacsony lesz, hogy ettől eltekinthetünk. Ennek hátránya, hogy léteznek olyan élek, melyeket az egyik oldalon kettéosztunk, a másikon viszont nem, mert az a két háromszög, melynek az adott él a közös éle, pont olyan távolságban vannak a találattól, hogy az egyiket felbontjuk, a másikat viszont nem. Ilyenkor kialakul egy olyan csúcs, amely felezi ezt az élet, viszont csak a feldarabolt háromszög irányában tagja a hálónak. Mivel a deformáció ezt az új csúcsot is elmozdítja bizonyos mértékben, így kialakulhatnak lyukak a hálóban. Szerencsére, ha megfelelően nagynak választjuk meg a határtávolságot, akkor ezeknek a lyukaknak a mérete elég elenyésző lesz ahhoz, hogy ne legyenek észrevehetőek, hiszen a csúcs elmozdulásának mértéke a deformáció során annál kisebb, minél távolabb van a csúcs a találattól. Ha pedig ezt sem szeretnénk, lehetőségünk van arra is, hogy minden esetben a teljes modellt bontsuk fel, ilyenkor ugyan sokkal több háromszöget generálunk, viszont a lyukak meg fognak szűnni. Továbbá az is lehetséges, hogy csak a súlyvonalak segítségével bontsuk fel a háromszögeket, melyek nem generálnak lyukakat a hálóban.

3.4.2.2 Felbontás a súlyvonalak mentén

A háromszög súlyvonalai a háromszög csúcsait a velük szemben lévő oldalak felezőpontjával kötik össze, a 3.4.2.2 ábrán látható módon. [27]



3.4.2.2. ábra: Háromszög súlyvonalai

A háromszög súlypontja a súlyvonalak metszéspontja. A háromszöget a háromszög csúcsait a súlyponttal összekötő egyenesek mentén daraboljuk fel. Ennek előnye, hogy nem alakulhatnak ki lyukak a hálóban, ami viszont egyúttal azzal is jár, hogy a kiinduló háromszög élei nem lesznek feldarabolva, így a háromszögek felbontása nem lesz homogén, tehát amikor a találat az egyik ilyen élet találja el (vagy nagyon közel kerül hozzá), és az él mindkét végpontja elég messze van a találat helyétől, akkor a két végpont csak kis mértékben fog elmozdulni, így a valósághoz nem túl közeli eredményt produkálva, hiszen a szóban forgó él nem fog tudni behorpadni.

A súlyvonalak mentén történő feldarabolás esetében a távolsági limit mérete függ attól, hogy hányadik iterációnál tartunk, de itt is meg van adva egy limit érték, amelynél ha kisebb a háromszög távolsága a becsapódás pontjától (ebben az esetben a távolság a találat és az ahhoz legközelebbi háromszögcsúcs távolsága), akkor a háromszög

felosztásra kerül. Ennek az értéknek van egy abszolút limitje is, de alapvetően ez egy dinamikus érték, amely függ a darabolási iteráció aktuális értékétől.

Kísérletezés után meghatároztam, hogy mely iteráció során nagyjából milyen távolsági limitet érdemes megszabni, így végül az alábbi hozzárendelést kaptam:

```
if (smallestDistance < Mathf.Max(0.3f * Mathf.Pow(2.72f, 0.6931f * (timesToBarycentricSubdivide - currentDepth)), 2.5f))
```

A smallestDistance a találat és az ahhoz legközelebb található csúcs távolsága, a 2.5f pedig az az érték, amelynél közelebbi háromszögeket mindenképpen feldarabol az algoritmus, amennyiben a területe nagyobb a területi limitnél. A timesToBarycentricSubdivide azt határozza meg, hogy összesen hányszor iteráljunk végig a feldaraboláson, a currentDepth pedig a jelenlegi iterációszámot határozza meg.

A kódban található függvényt a kísérletezés utáni értékekre illesztett trendvonal alapján határoztam meg.

3.4.2.3 A megfelelő darabolási módszer kiválasztása

Alapvetően az eddig felsorolt adatok alapján érdemes először a középvonalak mentén feldarabolni a háromszögeket, aztán pedig a súlyvonalak mentén.

3.4.3 Deformáció megvalósítása

Miután megfelelő módon feldaraboltuk a háromszögeket, egy elég sűrű háromszöghálót biztosítva, ezután a deformáció megvalósításához szükséges a háló csúcsainak olyan módú elmozdítása, hogy a végeredményként kialakuló mélyedés valóságszerű legyen. A deformáció mértéke függ a becsapódó lövedék sebességének nagyságától és irányától, a becsapódáspont és az adott csúcs pozíciójának távolságától, valamint a páncél vastagságától. Ezek alapján a következő összefüggésre jutottam:

```
//Use global coordinates
Vector3 worldPosition = transform.TransformPoint(verts[i]);

distance = Vector3.Distance(position, worldPosition);
//Avoid infinite depth holes
if(distance <= 0.2f)
{
worldPosition.x += velocity.x * 0.15f / (strength * 0.4f);
worldPosition.y += velocity.y * 0.15f / (strength * 0.4f);
worldPosition.z += velocity.z * 0.15f / (strength * 0.4f);
}
else if (distance <= 100.0f)
{
//edit the vertices
```

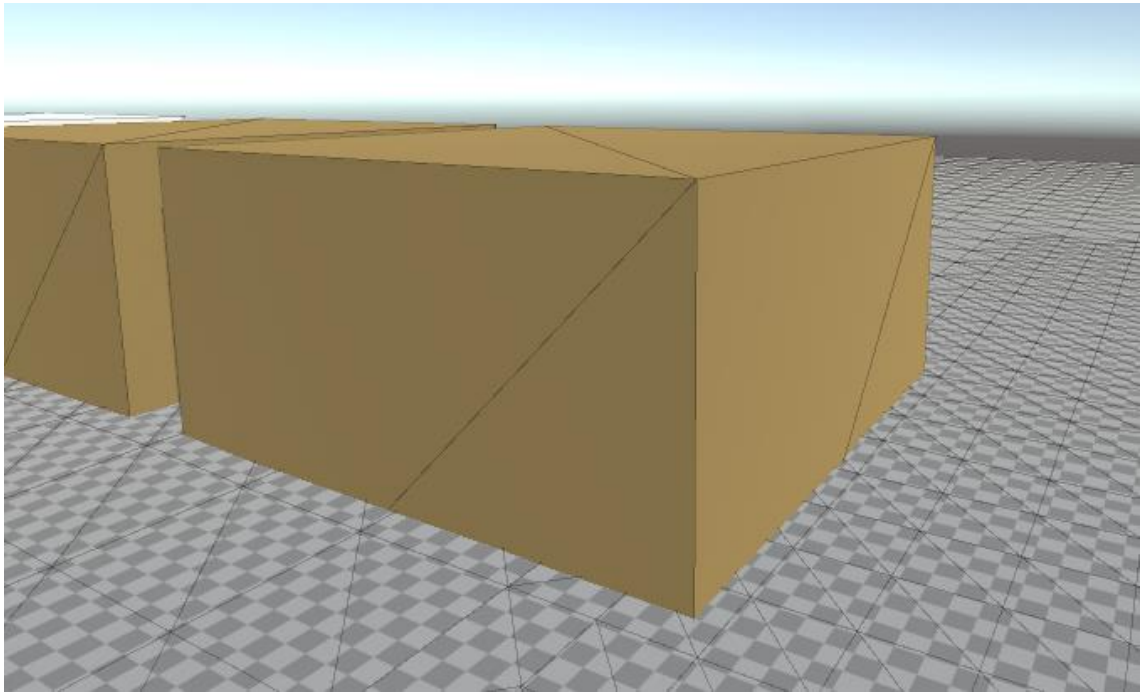
```
     worldPosition.x += velocity.x / (distance * distance * strength) *
0.150f;
     worldPosition.y += velocity.y / (distance * distance * strength) *
0.150f;
     worldPosition.z += velocity.z / (distance * distance * strength) *
0.150f;
     }

     //Change back to local coordinates
     verts[i] = transform.InverseTransformPoint(worldPosition);
```

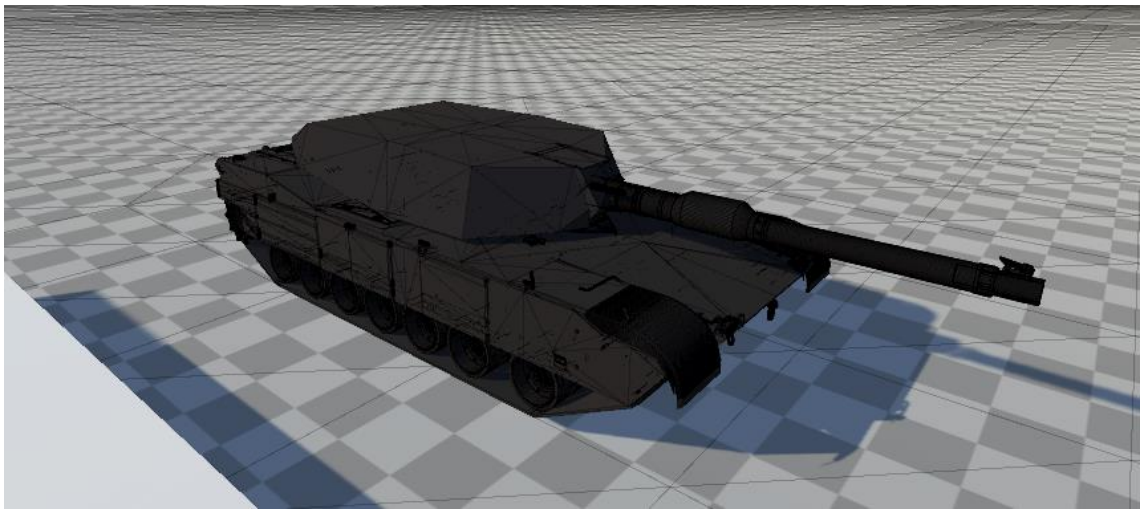
Amennyiben a csúcs és a becsapódáspont távolsága nagyon kicsi, akkor a kráter mélysége túlságosan nagy lesz a képlet alapján, így létrehoztam egy limitet, amelynél kisebb távolság esetén az elmozdulás mértéke nem függ ettől.

3.4.4 Deformáció végeredménye

A kiindulási háromszöghálókat a 3.4.4-1. és 3.4.4-2. ábrákon láthatjuk.



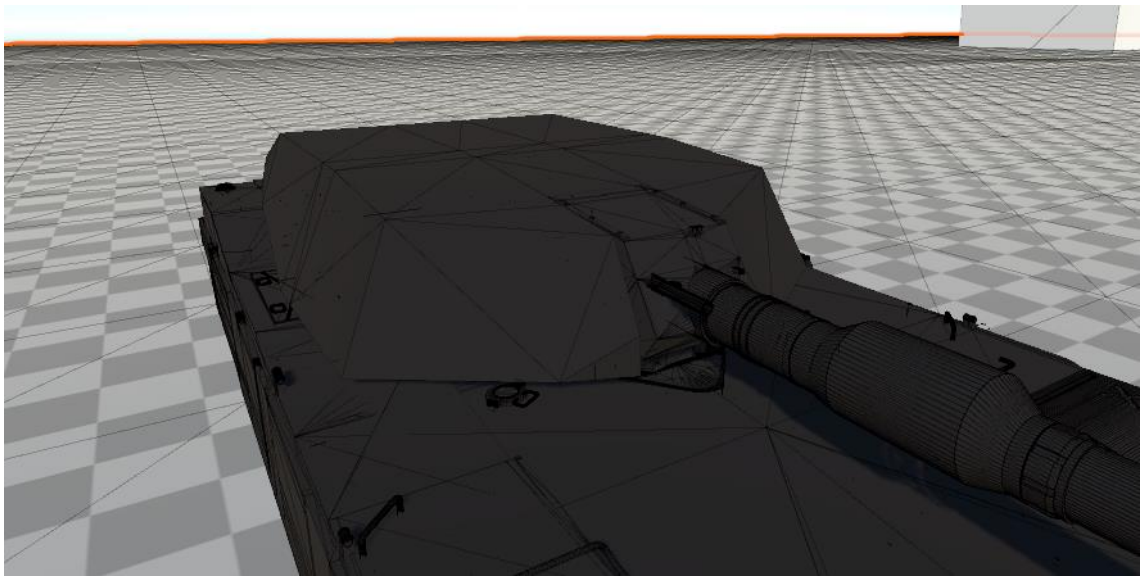
3.4.4-1. ábra: Téglatest kiinduló hálója



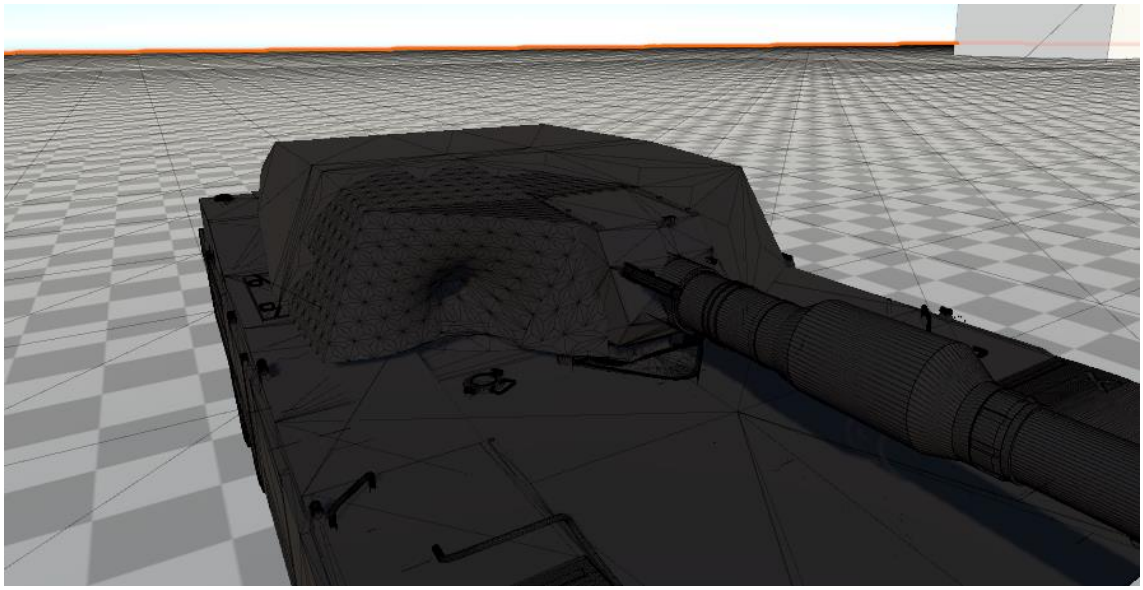
3.4.4-2. ábra: Abrams kiinduló hálója

Összességében az algoritmus sebessége kifogástalan, mindössze akkor fordulhat elő kisebb akadás futás közben, amikor több találat után próbálunk ismét meglőni egy olyan tárgyat, melynek nem homogén a páncélja. Tulajdonképpen ekkor sem a darabolóalgoritmus miatt történik lassulás, hanem azért, mert az eltalált háromszöget kereső függvénynek jelentősen több elemen kell végigiterálnia, mint a korábbi esetekben. Ennek elkerülése érdekében érdemes több kisebb testből összeállítani a modelleket, esetleg úgy megváltoztatni a működést, hogy amennyiben egy test egy adott értéknél több találatot kap, akkor semmisüljön meg.

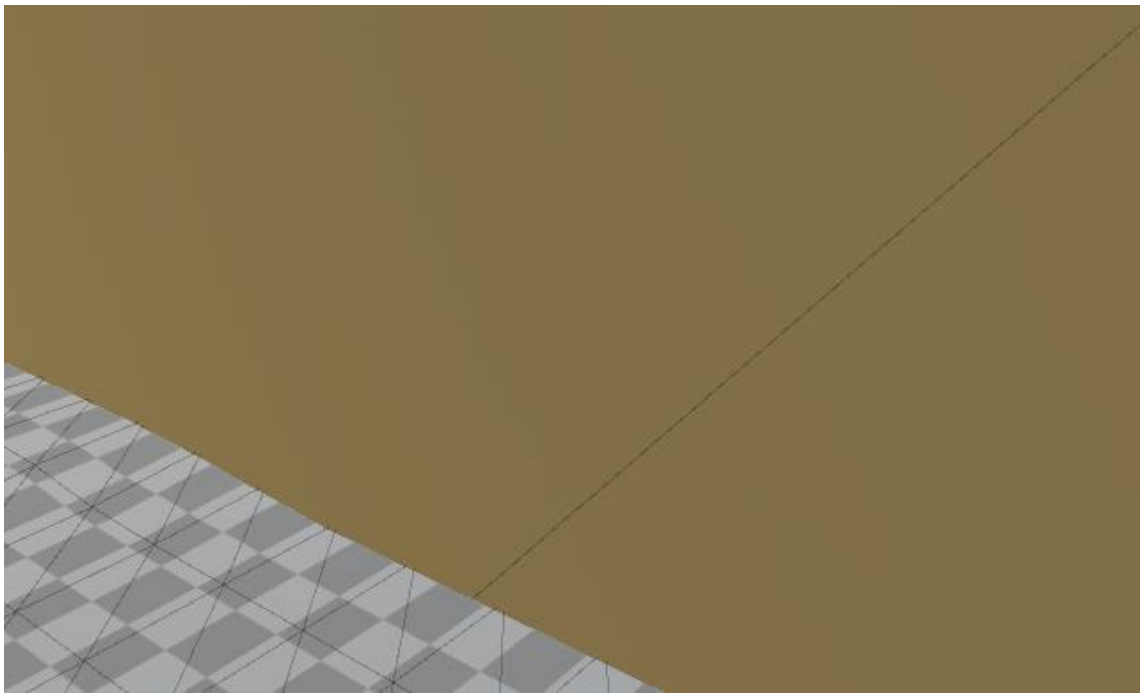
Néhány előtte-utána fotó a becsapódásokról, a 3.4.4-3.és 3.4.4-4., valamint a 3.4.4-5., 3.4.4-6. ábrákon. A 3.4.4-3.és 3.4.4-4. ábrákon nagyon jól kivehető, hogy viszonylag összetettebb hálókon milyen szépen darabol adott területen az algoritmus. A 3.4.4-5.és 3.4.4-6. ábrákon pedig azt láthatjuk, hogy viszonylag alacsonyabb felbontási részletesség mellett is alapvetően jó minőségű deformációkat kapunk.



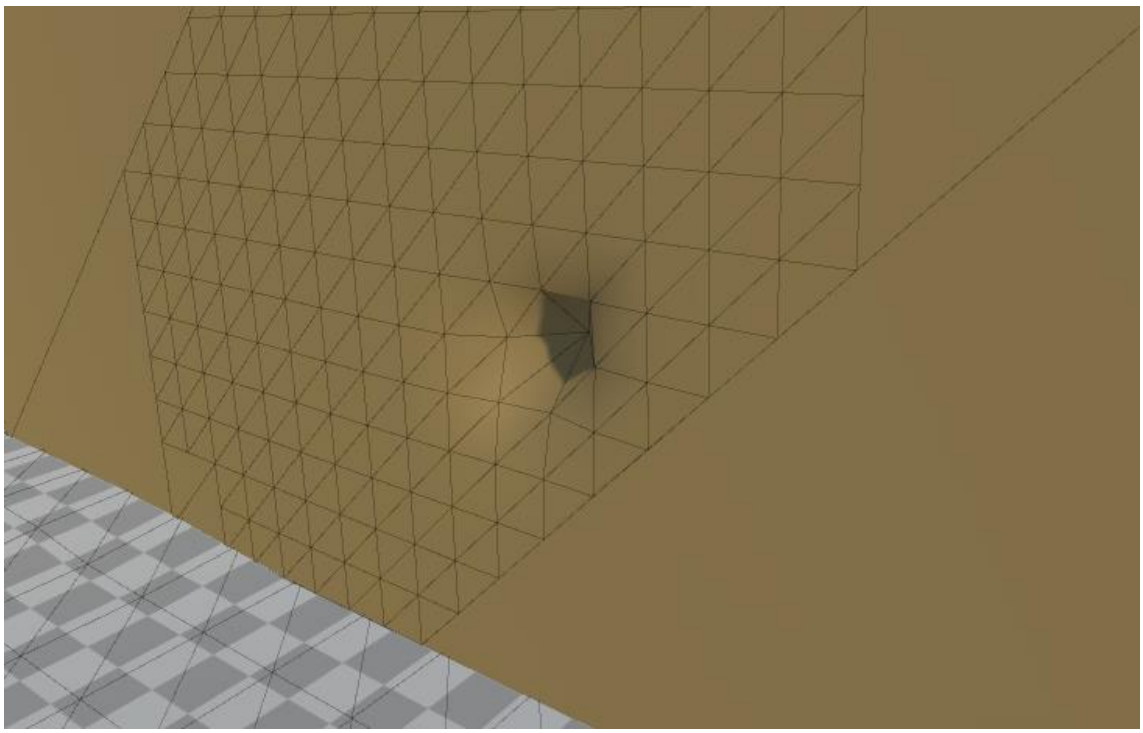
3.4.4-3. ábra: Abrams találat előtt



3.4.4-4. ábra: Abrams találat után

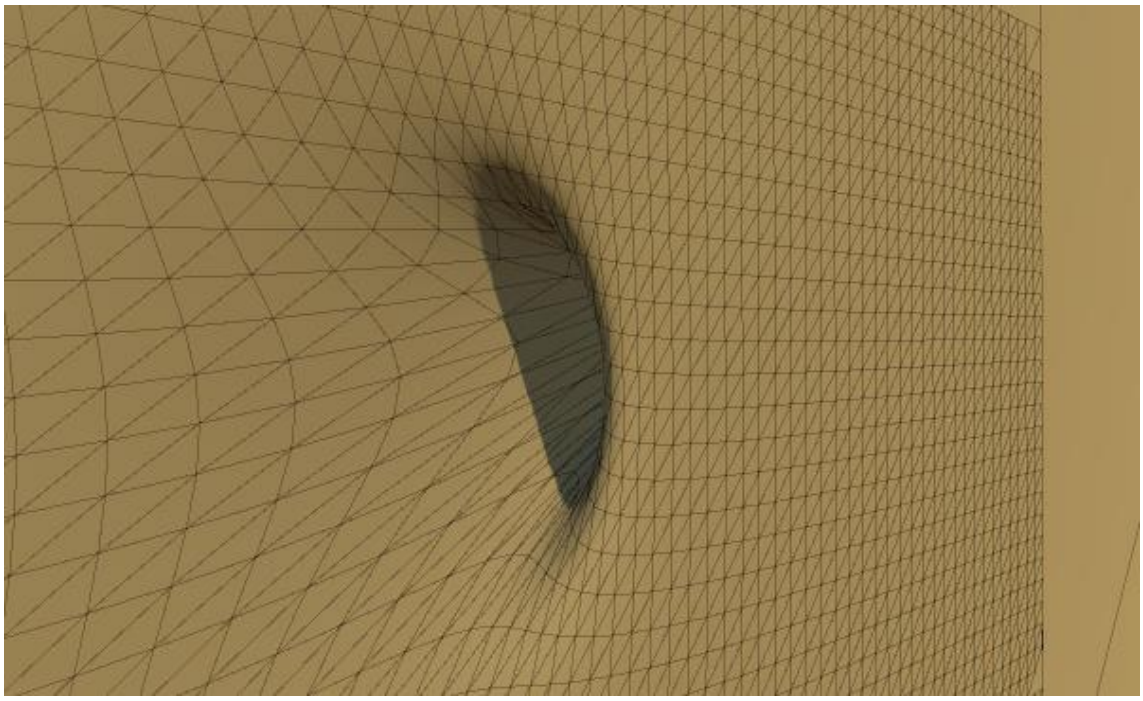


3.4.4-5. ábra: Téglatest találat előtt



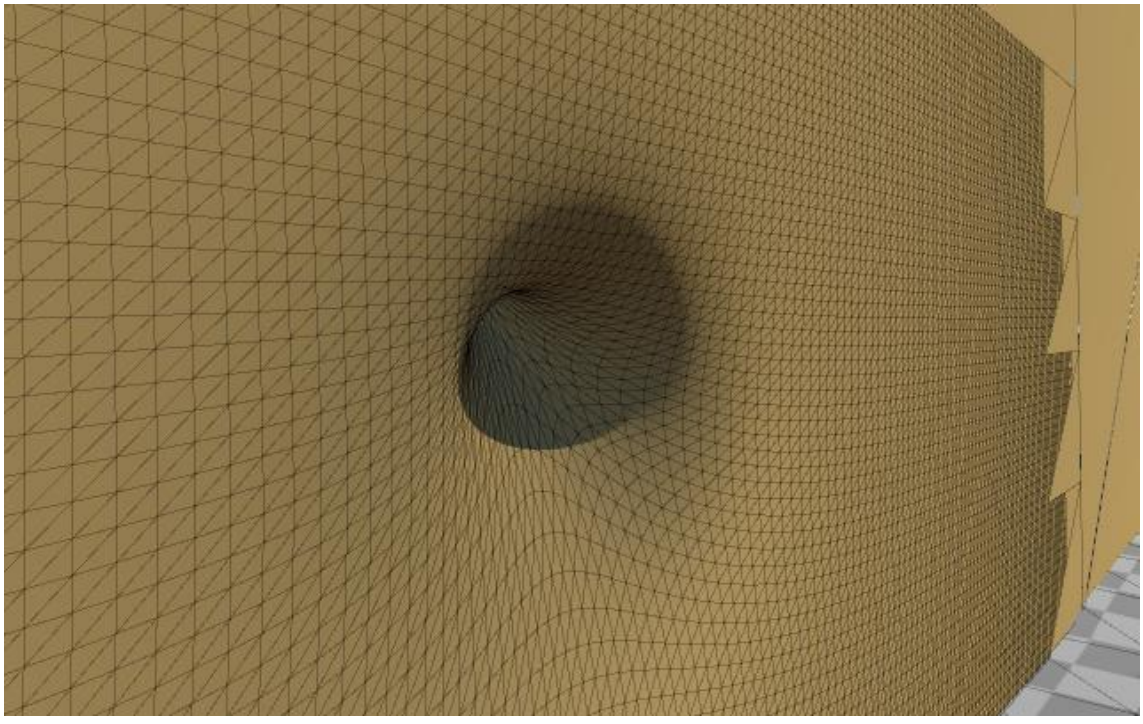
3.4.4-6. ábra: Téglatest találat után

A 3.4.4-7. ábrán láthatunk példát egy viszonylag nagy beesési szögű találatra. Ahogyan az a képen is látszik, a mélyedés iránya megegyezik a beérkezett lövedék irányával.



3.4.4-7. ábra: Nagy beesési szögben érkező lövedék által okozott deformáció

A 3.4.4-8. ábrán példát láthatunk egy nagy minőségben felbontott háló deformációjára. Látható, hogy ebben az esetben a végeredmény sokkal élethűbb.



3.4.4-7. ábra: Nagy felbontású háló deformációja

4 Értékelés és továbbfejlesztési lehetőségek

4.1 Értékelés

Összességében, szerintem a feladat legtöbb részét sikerült megfelelő részletességgel és minőséggel megoldani.

A lánctalpak és a felfüggesztés viselkedése terepen kifejezetten valóságos lett, viszont sajnos a meghajtás működése nem lett tökéletes két kisebb hiba miatt. Egyrészt az egész jármű balra akar csúszni a felfüggesztésen, ezért a test oldalirányú pozícióját rögzítenem kellett. Másrészt pedig a lánctalpak nem forognak hátrafelé, annak ellenére, hogy a meghajtó görgő igen. Látszik, hogy a görgő fejt ki erőt a lánctalpra, mert az elkezd lassulni és megnyúlni, amikor hátrafelé próbálunk haladni. Ennek oka feltételezhetően a modell hierarchiájában keresendő. Amennyiben a jármű Rigidbody elemeit nem szülő gyerek viszonyba rendezem, hanem egy szintre hozom őket, akkor a csúszó viselkedés elmúlik. Mindemellett a hidropneumatikus felfüggesztés viselkedése kifejezetten élethű lett.

A lövedéktípusok mechanikáját alapvetően hasonló témával foglalkozó számítógépes játékokból merítettem, ezek működése nagy mértékben hasonlít az azokban megvalósított módszerekhez.

A lövedékbecsapódások szimulációja szerintem kifejezetten látványos, az ehhez megalkotott algoritmus pedig viszonylag hatékony, mindemellett pedig nagy mértékben testreszabható lett.

4.2 Továbbfejlesztési lehetőségek

A projektet több irányban lehet továbbfejleszteni.

Egy lehetséges irány a valósághűség felé hajlik, ahol valós katonai szituációk modellezésére, azok virtuális gyakorlására lehetne használni a projektet. Ehhez ki kell alakítani a megfelelő terepadottságokat, valamint érdemes kibővíteni a támogatott egységtípusok számát, esetleg további járművekkel, gyalogos katonákkal.

Egy másik opció egy számítógépes játék kialakítása, amely alapvetően inkább játékélményt javító változtatásokat igényel, többjátékos játék esetén egy megfelelő multiplayer mód kialakításával.

Irodalomjegyzék

[1] Tank and AFV News: *Photo of the Day: Hydro-Abrams*, https://tankandafvnews.com/2017/05/15/photo-of-the-day-hydro-abrams/, 2017.05.17

[2] Wikipedia: *Hydropneumatic suspension*, https://en.wikipedia.org/wiki/Hydropneumatic_suspension, 2021.11.17.

[3] Wikipedia: *Kinetic energy penetrator*, https://en.wikipedia.org/wiki/Kinetic_energy_penetrator, 2021.12.09.

[4] Wikipedia: Sloped armour, https://en.wikipedia.org/wiki/Sloped_armour, 2021.09.01.

[5] Wikipedia: *German encounter of Soviet T-34 and KV tanks*, https://en.wikipedia.org/wiki/German_encounter_of_Soviet_T-34_and_KV_tanks, 2021.06.26.

[6] Wikipedia: *Ricochet*, https://en.wikipedia.org/wiki/Ricochet, 2021.11.30.

[7] Turbosquid: *3D Abrams Free*, https://www.turbosquid.com/3d-models/3d-m1a1-abrams-1619102

[8] Unity Documentation: *Rigidbody*, https://docs.unity3d.com/Manual/class-Rigidbody.html, 2021.12.03.

[9] Unity Documentation: *Mesh Collider*, https://docs.unity3d.com/Manual/class-MeshCollider.html, 2021.12.03.

[10] Unity Documentation: *Wheel Collider*, https://docs.unity3d.com/Manual/class-WheelCollider.html, 2021.12.03.

[11] Unity Documentation: *Spring Joint*, https://docs.unity3d.com/Manual/class-SpringJoint.html, 2021.12.03.

[12] Unity Documentation: *Hinge Joint*, https://docs.unity3d.com/Manual/class-HingeJoint.html, 2021.12.03.

[13] Wikipedia: *Third-person shooter*, https://hu.wikipedia.org/wiki/Third-person_shooter, 2020.11.06.

[14] Unity Documentation: *Rotation and Orientation in Unity*, https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html, 2021.12.03.

[15] Unity Documentation: *Physics in Unity 5.0*, https://docs.unity3d.com/Manual/UpgradeGuide5-Physics.html, 2021.12.03.

[16] Unity Documentation: *Joints*, https://docs.unity3d.com/Manual/Joints.html, 2021.12.03.

[17] Unity Documentation: *Project Settings*, https://docs.unity3d.com/Manual/comp-ManagerGroup.html, 2021.12.03.

[18] Unity Documentation: *MonoBehaviour.OnCollisionEnter(Collision)*, https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnCollisionEnter.html 2021.12.03.

[19] Wikipedia: *Recoil*, https://hu.wikipedia.org/wiki/Recoil, 2021.11.20.

[20] Unity Documentation: *Particle systems*, *https://docs.unity3d.com/Manual/ParticleSystems.html*, 2021.12.03.

[21] Unity Asset Store: *Unity Particle Pack*, https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-particle-pack-127325#description, 2021.06.10.

[22] Wikipedia: *Polygon mesh*, https://en.wikipedia.org/wiki/Polygon_mesh, 2021.11.20.

[23] Unity Documentation: *Mesh*, *https://docs.unity3d.com/ScriptReference/Mesh.html*, 2021.12.03.

[24] Github Gist: TriangleArea.cs, *https://gist.github.com/openroomxyz/c67e77f710eb53aaaae7390ded9ce86d*, 2020.04.01.

[25] Wikipedia: *Középvonal,* https://hu.wikipedia.org/wiki/K%C3%B6z%C3%A9pvonal, 2021.02.08.

[26] Unity Answers: *Does anyone have any code to subdivide a mesh and keep vertices shared?*, https://answers.unity.com/questions/259127/does-anyone-have-any-code-to-subdivide-a-mesh-and.html, 2012.05.29.

[27] Wikipedia: *Középvonal,* https://hu.wikipedia.org/wiki/S%C3%BAlyvonal, 2021.06.30.