RimWorld Damage Mechanic

1. VANILLA

2. CE

Generic(1)

1. Verb, projectile and damage are different things;
2. Melee verbs are defined as <tools>, each with its respective capacities. The capacities in ToolCapacityDef is used in ManeuverDef for its verbClass.

   Conclusion : Melee = capacities + power + cooldownTime (Armor Penetration uses different formula)

```
<ThingDef ParentName="BaseHumanMakeableGun">
  <defName>Gun_ChargeRifle</defName>
  <label>charge rif
  <description>A ch
```

```
<tools>
  <li>
    <label>stock</label>
    <capacities>
      <li>Blunt</li>
    </capacities>
    <power>9</power>
    <cooldownTime>2</cooldownTime>
  </li>
  <li>
    <label>barrel</label>
    <capacities>
      <li>Blunt</li>
      <li>Poke</li>
    </capacities>
    <power>9</power>
    <cooldownTime>2</cooldownTime>
  </li>
</tools>
```

```
<ManeuverDef>
  <defName>Poke</defName>
  <requiredCapacity>Poke</requiredCapacity>
  <verb>
    <verbClass>Verb_MeleeAttackDamage</verbClass>
    <meleeDamageDef>Poke</meleeDamageDef>
  </verb>
  <logEntryDef>MeleeAttack</logEntryDef>
  <combatLogRulesHit>Maneuver_Poke_MeleeHit</combatLogRulesHit>
  <combatLogRulesDeflect>Maneuver_Poke_MeleeDeflect</combatLogRulesDeflect>
  <combatLogRulesMiss>Maneuver_Poke_MeleeMiss</combatLogRulesMiss>
  <combatLogRulesDodge>Maneuver_Poke_MeleeDodge</combatLogRulesDodge>
</ManeuverDef>

<ManeuverDef>
  <defName>Smash</defName>
  <requiredCapacity>Blunt</requiredCapacity>
  <verb>
    <verbClass>Verb_MeleeAttackDamage</verbClass>
    <meleeDamageDef>Blunt</meleeDamageDef>
  </verb>
  <logEntryDef>MeleeAttack</logEntryDef>
  <combatLogRulesHit>Maneuver_Smash_MeleeHit</combatLogRulesHit>
  <combatLogRulesDeflect>Maneuver_Smash_MeleeDeflect</combatLogRulesDeflect>
  <combatLogRulesMiss>Maneuver_Smash_MeleeMiss</combatLogRulesMiss>
  <combatLogRulesDodge>Maneuver_Smash_MeleeDodge</combatLogRulesDodge>
</ManeuverDef>
```

```
<ToolCapacityDef>
  <defName>Blunt</defName>
  <label>blunt</label>
</ToolCapacityDef>
```

```
<ToolCapacityDef>
  <defName>Poke</defName>
  <label>poke</label>
</ToolCapacityDef>
```

```
    <muzzleFlashScale>9</muzzleFlashScale>
  </li>
</verbs>
```

Generic(2)

- Damage Calculation：
  1. Attacker calculates the final out damage and pass the Damageinfo to its victim;
  2. The victim calls Thing.TakeDamage, in that method:
     1. Applies damageMultipliers (i.e. Walls receive 2x damage from Bomb & Thump)
     2. PreApplyDamage, where the damage gets adjusted before damageWorker.Apply
        - GeneDef.damageFactors has the highest priority (applies before shield)

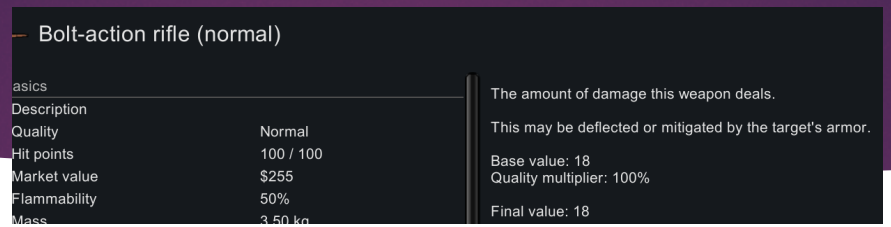
          In vanilla, only Fire Resistant & Fire Weakness gene fall into this category

```
<damageMultipliers>
  <li>
    <damageDef>Bomb</damageDef>
    <multiplier>2</multiplier>
  </li>
  <li>
    <damageDef>Thump</damageDef>
    <multiplier>2</multiplier>
  </li>
</damageMultipliers>
```

```
public override void PreApplyDamage(ref DamageInfo dinfo, out bool absorbed)
{
    if (ModsConfig.BiotechActive && genes != null)
    {
        float num = genes.FactorForDamage(dinfo);
        if (num != 1f)
        {
            dinfo.SetAmount(dinfo.Amount * num);
        }
    }
    base.PreApplyDamage(ref dinfo, out absorbed);
    if (!absorbed)
    {
        health.PreApplyDamage(dinfo, out absorbed);
    }
}
```

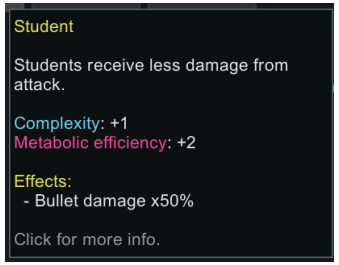
Test

```
public float energyLossPerDamage = 0.033f;
```



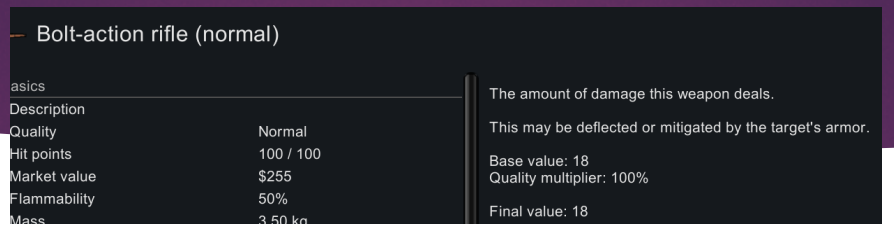
- 18 x 0.033 = 0.594
- 110% - 59.4% = 51%

- 18 x 0.033 x 0.5 = 0.297
- 110% - 29.7% = 80%



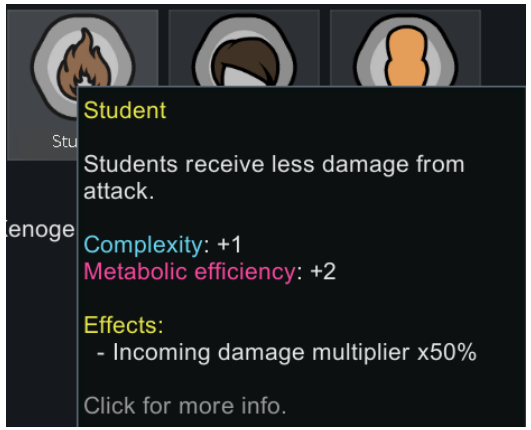
Test

```
public float energyLossPerDamage = 0.033f;
```



- 18 x 0.033 = 0.594
- 110% - 59.4% = 51%

- 18 x 0.033 = 0.594
- 110% - 59.4% = 51%



Generic(2)

```xml
<damageMultipliers>
  <li>
    <damageDef>Bomb</damageDef>
    <multiplier>2</multiplier>
  </li>
  <li>
    <damageDef>Thump</damageDef>
    <multiplier>2</multiplier>
  </li>
</damageMultipliers>
```

- Damage Calculation：
  1. Attacker calculates the final out damage and pass the Damageinfo to its victim;
  2. The victim calls Thing.TakeDamage, in that method:
     1. Applies damageMultipliers (i.e. Walls receive 2x damage from Bomb & Thump)
     2. PreApplyDamage, where the damage gets adjusted before damageWorker.Apply
        - GeneDef.damageFactors has the highest priority (applies before shield)

          In vanilla, only Fire Resistant & Fire Weakness gene fall into this category
     3. damageWorker.Apply, where armor reduction and other factors get applied

```
public override void PreApplyDamage(ref DamageInfo dinfo, out bool absorbed)
{
    if (ModsConfig.BiotechActive && genes != null)
    {
        float num = genes.FactorForDamage(dinfo);
        if (num != 1f)
        {
            dinfo.SetAmount(dinfo.Amount * num);
        }
    }
    base.PreApplyDamage(ref dinfo, out absorbed);
    if (!absorbed)
    {
        health.PreApplyDamage(dinfo, out absorbed);
    }
}
```

Trait ”tough”，gene ”robust”、”delicate”

```
if (dinfo.Def.ExternalViolenceFor(pawn))
{
    num *= pawn.GetStatValue(StatDefOf.IncomingDamageFactor);
}
```

- Adjust stat <IncomingDamageFactor>, only applies to externalViolence

  There’s no damageDef in vanilla whose externalViolence is false => ~~Self hurting also counts as externalViolence~~

- Applies **after** armor reduction

Vanilla

```
damageInfo.SetBodyRegion(BodyPartHeight.Undefined, BodyPartDepth.Outside);
```

- Most Melee attack has no actual part when inited, only the height and the depth of the part
  - You could hardly hit organs with melee

- Bullet has no body region at all, so where it will hit is totally random

```
protected virtual BodyPartRecord ChooseHitPart(DamageInfo dinfo, Pawn pawn)
{
    return pawn.health.hediffSet.GetRandomNotMissingPart(dinfo.Def, dinfo.Height, dinfo.Depth);
}
```

Melee DamageWorker

- Cut, Blunt, Scratch, Bite

```
protected override BodyPartRecord ChooseHitPart(DamageInfo dinfo, Pawn pawn)
{
    return pawn.health.hediffSet.GetRandomNotMissingPart(dinfo.Def, dinfo.Height, BodyPartDepth.Outside);
}
```

  - Outer Bodypart only

- Stab

```
protected override BodyPartRecord ChooseHitPart(DamageInfo dinfo, Pawn pawn)
{
    BodyPartRecord randomNotMissingPart = pawn.health.hediffSet.GetRandomNotMissingPart(dinfo.Def, dinfo.Height, dinfo.Depth);
    if (randomNotMissingPart.depth != BodyPartDepth.Inside && Rand.Chance(def.stabChanceOfForcedInternal))
    {
        BodyPartRecord randomNotMissingPart2 = pawn.health.hediffSet.GetRandomNotMissingPart(dinfo.Def, BodyPartHeight.Undefined, BodyPartDepth.Inside, randomNotMissingPart);
        if (randomNotMissingPart2 != null)
        {
            return randomNotMissingPart2;
        }
    }
    return randomNotMissingPart;
}
```

  - <stabChanceOfForcedInternal> defines the probability to hit inside

Conclusion

- Bodypart to hit：
  - Melee: most of the time outer part, only stab has stabChanceOfForcedInterna
  - Ranged: totally random
- Damage Calculation：
  - Attacker calculates damage output, pass that to its victim
  - Victim：
    1. Applies <damageFactors> if such gene exists
    2. PreApplyDamage, where the shield absorbs damage
    3. damageWorkre.Apply:
       1. Damage propagation (frag with 50 dmg, deals 25 dmg each to left arm and leg)
       2. ApplyDamageToPart (processes armor reduction, then applies global damage reduction (IncomingDamageFactor) if it's "externalViolence")
       3. FinalizeAndAddInjury (Add damage hediff)

CE

- CE reroutes vanilla armor reduction method by using transpiler:

```
private static void ArmorReroute(Pawn pawn, ref DamageInfo dinfo, out bool deflectedByArmor, out bool diminishedByArmor)
{
    var newDinfo = ArmorUtilityCE.GetAfterArmorDamage(dinfo, pawn, dinfo.HitPart, out deflectedByArmor, out diminishedByArmor, out shieldAbsorbed);
    if (dinfo.HitPart != newDinfo.HitPart)
    {
        if (pawn.Spawned)
        {
            LessonAutoActivator.TeachOpportunity(CE_ConceptDefOf.CE_ArmorSystem, OpportunityType.Critical);    // Inform the player about armor deflection
        }
    }
    Patch_CheckDuplicateDamageToOuterParts.lastHitPartHealth = pawn.health.hediffSet.GetPartHealth(newDinfo.HitPart);

    dinfo = newDinfo;
}
```

- All the damage calculation happens inside ArmorUtilityCE.GetAfterArmorDamage

ArmorUtilityCE.GetAfterArmorDamage

- 4 steps:
  1. Ambient damage (fire & electricity) -> percentage reduction -> return

```
// In case of ambient damage (fire, electricity) we apply a percentage reduction formula based on the sum of all applicable armor
if (isAmbientDamage)
{
    dinfo.SetAmount(Mathf.CeilToInt(GetAmbientPostArmorDamage(dmgAmount, originalDinfo.Def.armorCategory.armorRatingStat, pawn, hitPart)));
    armorDeflected = dinfo.Amount <= 0;
    return dinfo;
}
else if (deflectionComp != null)...
```

```
float ArmorUtilityCE.GetAmbientPostArmorDamage(float dmgAmount, StatDef armorRatingStat, Pawn pawn, BodyPartRecord part)
Calculates damage reduction for ambient damage types (fire, electricity) versus natural and worn armor of a pawn. Adds up the total armor percentage (clamped at 0-100%) and multiplies damage by that amount.
返回结果:
  The post-armor damage ranging from 0 to the original amount
```

ArmorUtilityCE.GetAfterArmorDamage

- 4 steps:
  1. Ambient damage (fire & electricity)
     -> percentage reduction
     -> return
  2. Applies shield, then calculates from outer armor to inside

```
// Apply worn armor
if (involveArmor && pawn.apparel != null && !pawn.apparel.WornApparel.NullOrEmpty())
{
    var apparel = pawn.apparel.WornApparel;

    // Check for shields first
    var shield = apparel.FirstOrDefault(x => x is Apparel_Shield);
    if (shield != null)
    {
        // Determine whether the hit is blocked by the shield
        var blockedByShield = false;
        if (!(dinfo.Weapon?.IsMeleeWeapon ?? false))
        {
            var shieldDef = shield.def.GetModExtension<ShieldDefExtension>();
            if (shieldDef == null)
            {
                Log.ErrorOnce("Combat Extended :: shield " + shield.def.ToString() + " is Apparel_Shield but has no ShieldDefExtension", shield.def.GetHashCode() + 12748102);
            }
            else
            {
                var hasCoverage = shieldDef.PartIsCoveredByShield(hitPart, pawn);
                if (hasCoverage)
                {
                    // Right arm is vulnerable during warmup/attack/cooldown
                    blockedByShield = !((pawn.stances?.curStance as Stance_Busy)?.verb != null && hitPart.IsInGroup(CE_BodyPartGroupDefOf.RightArm));
                }
            }
        }
        // Try to penetrate the shield
        if (blockedByShield && !TryPenetrateArmor(dinfo.Def, shield.GetStatValue(dinfo.Def.armorCategory.armorRatingStat), ref penAmount, ref dmgAmount, shield))
        {
            shieldAbsorbed = true;
            armorDeflected = true;
            dinfo.SetAmount(0);

            // Apply secondary damage to shield
            if (dinfo.Weapon?.projectile is ProjectilePropertiesCE props && !props.secondaryDamage.NullOrEmpty())
            {
                foreach (var sec in props.secondaryDamage)
                {
                    if (shield.Destroyed || !Rand.Chance(sec.chance))
                    {
                        break;
                    }
                    var secDinfo = sec.GetDinfo();
                    var pen = secDinfo.ArmorPenetrationInt; //GetPenetrationValue(originalDinfo);
                    var dmg = (float)secDinfo.Amount;
                    TryPenetrateArmor(secDinfo.Def, shield.GetStatValue(secDinfo.Def.armorCategory.armorRatingStat), ref pen, ref dmg, shield);
                }
            }

            return dinfo;
        }
    }
```

ArmorUtilityCE.GetAfterArmorDamage

- 4 steps:
  1. Ambient damage (fire & electricity)
     -> percentage reduction
     -> return
  2. Applies shield, then calculates from outer armor to inside

```
// Apparel is arranged in draw order, we run through reverse to go from Shell -> OnSkin
for (var i = apparel.Count - 1; i >= 0; i--)
{
    var app = apparel[i];
    if (app != null
            && app.def.apparel.CoversBodyPart(hitPart)
            && !TryPenetrateArmor(dinfo.Def, app.PartialStat(dinfo.Def.armorCategory.armorRatingStat, hitPart), ref penAmount, ref dmgAmount, app))
    {
        if (dinfo.Def.armorCategory.armorRatingStat == StatDefOf.ArmorRating_Sharp)
        {
            if (deflectionComp != null)
            {
                deflectionComp.deflectedSharp = true;
            }
        }

        // Hit was deflected, convert damage type
        //armorReduced = true;
        dinfo = GetDeflectDamageInfo(dinfo, hitPart, ref dmgAmount, ref penAmount);
        if (app == apparel.ElementAtOrDefault(i))   //Check whether the "deflecting" apparel is still in the WornApparel - if not, the next loop checks aga
        {
            i++;    // We apply this piece of apparel twice on conversion, this means we can't use deflection on Blunt or else we get an infinite loop of e
        }
    }
    if (dmgAmount <= 0)
    {
        dinfo.SetAmount(0);
        armorDeflected = true;
        return dinfo;
    }
}
```

ArmorUtilityCE.GetAfterArmorDamage

- 4 steps:
  1. Ambient damage (fire & electricity)
     -> percentage reduction
     -> return
  2. Applies shield, then calculates from outer armor to inside
  3. Add hitparts, for damage that harm all layers, it will be all the parts until its outermost parent part

```
var partsToHit = new List<BodyPartRecord>()
{
    hitPart
};
if (dinfo.Def.harmAllLayersUntilOutside)
{
    var curPart = hitPart;
    while (curPart.parent != null && curPart.depth == BodyPartDepth.Inside)
    {
        curPart = curPart.parent;
        partsToHit.Add(curPart);
    }
}
```

ArmorUtilityCE.GetAfterArmorDamage

- 4 steps:
  1. Ambient damage (fire & electricity)
     -> percentage reduction
     -> return
  2. Applies shield, then calculates from outer armor to inside
  3. Tries to penetrate hitparts

```
var isSharp = dinfo.Def.armorCategory.armorRatingStat == StatDefOf.ArmorRating_Sharp;
var partDensityStat = isSharp
                    ? CE_StatDefOf.BodyPartSharpArmor
                    : CE_StatDefOf.BodyPartBluntArmor;
var partDensity = pawn.GetStatValue(partDensityStat);   // How much armor is provided by sheer meat
for (var i = partsToHit.Count - 1; i >= 0; i--)
{
    var curPart = partsToHit[i];
    var coveredByArmor = curPart.IsInGroup(CE_BodyPartGroupDefOf.CoveredByNaturalArmor);
    var armorAmount = coveredByArmor ? pawn.PartialStat(dinfo.Def.armorCategory.armorRatingStat, curPart, dmgAmount, penAmo

    // Only apply damage reduction when penetrating armored body parts
    if (!TryPenetrateArmor(dinfo.Def, armorAmount, ref penAmount, ref dmgAmount, null, partDensity))
    {
        dinfo.SetHitPart(curPart);
        if (isSharp && coveredByArmor)
        {
            if (dinfo.Def.armorCategory.armorRatingStat == StatDefOf.ArmorRating_Sharp)
            {
                if (deflectionComp != null)
                {
                    deflectionComp.deflectedSharp = true;

                    deflectionComp.weapon = originalDinfo.Weapon;
                }
            }
            // For Mechanoid natural armor, apply deflection and blunt armor
            dinfo = GetDeflectDamageInfo(dinfo, curPart, ref dmgAmount, ref penAmount);

            // Fetch armor rating stat again in case of deflection conversion to blunt
            TryPenetrateArmor(dinfo.Def, pawn.GetStatValue(dinfo.Def.armorCategory.armorRatingStat), ref penAmount, ref dmg
        }
        break;
    }
    if (dmgAmount <= 0)
    {
        dinfo.SetAmount(0);
        armorDeflected = true;
        return dinfo;
    }
}
```

ArmorUtilityCE.GetAfterArmorDamage

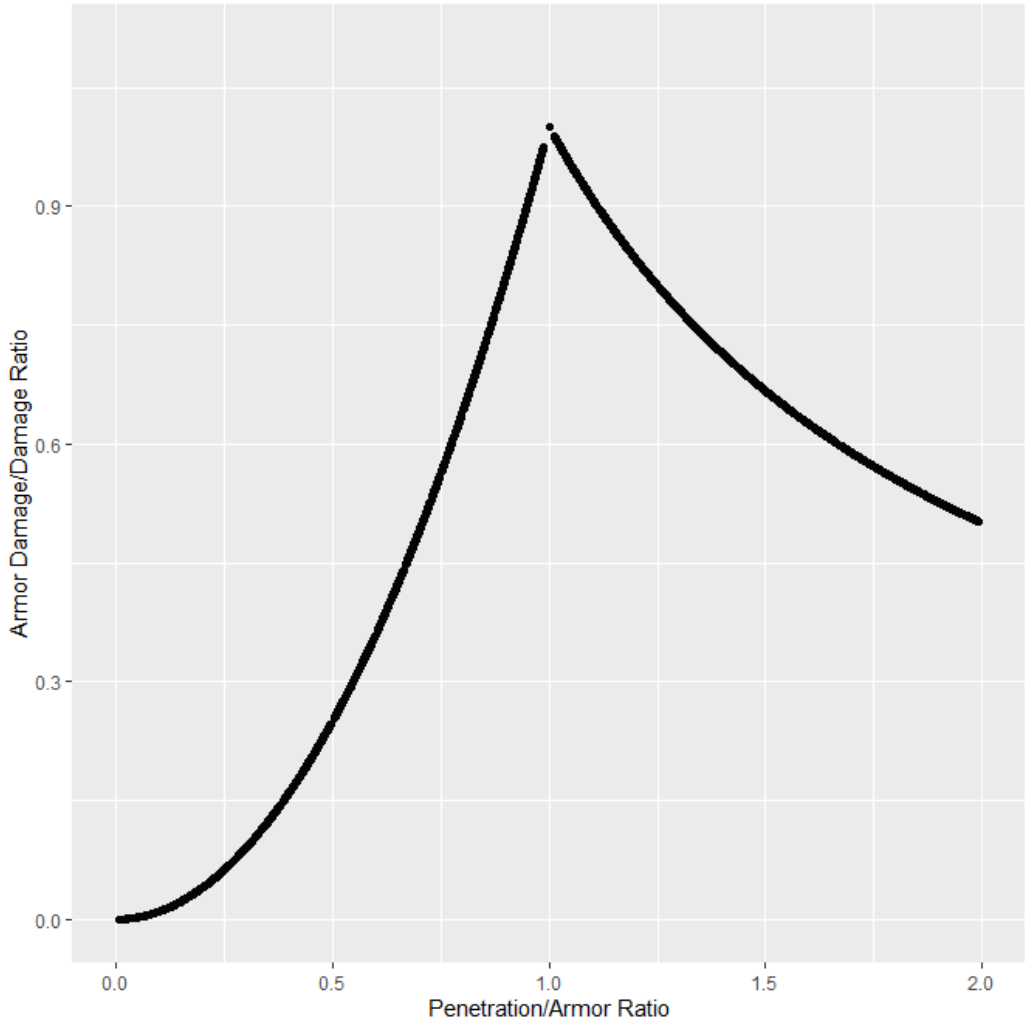
- 4 steps：
  1. Ambient damage (fire & electricity)
     -> percentage reduction -> return
  2. Applies shield, then calculates from outer armor to inside
  3. Tries to penetrate hitparts
  4. For partial penetration it will calculate another instance of damage, otherwise returns the damage info after all the possible reduction

```
// Applies blunt damage from partial penetrations.
if (isSharp && (dinfo.Amount > dmgAmount))
{
    pawn.TakeDamage(GetDeflectDamageInfo(dinfo, hitPart, ref dmgAmount, ref penAmount, true));
}
// Return damage info.
dinfo.SetAmount(dmgAmount);
return dinfo;
```

```
private static readonly StuffCategoryDef[] softStuffs = { StuffCategoryDefOf.Fabric, DefDatabase<StuffCategoryDef>.GetNamed("Leathery") };
```

Core Method：TryPenetrateArmor

- Things you may not know: Soft/Hard Armor
  - Soft Armor: all "armor" made of fabric and leathery stuff. Receives no damage from blunt damage, and could only receive up to 20% of the sharp damage it reduces
  - Hard Armor
    - Fully negates blunt damage when its penetration is less than half of the armor
    - Otherwise:
      - armorDamage = (dmgAmount - newDmgAmount) * Mathf.Min(1.0f, (penAmount * penAmount) / (armorAmount * armorAmount)) + newDmgAmount * Mathf.Clamp01(armorAmount / penAmount);
      - newDmgAmount = dmgAmount * (penAmount – armorAmount) / penAmount
    - theorem1：Armor receives the most damage from attack with equivalent penetration as the armor itself



Core Method: TryPenetrateArmor

- Other formula:
  - Sharp damage -> deflected when penAmount < armorAmount
  - newPenAmount -> penAmount - armorAmount
  - dmgMult -> newPenAmount / penAmount
  - newDmgAmount -> dmgAmount * dmgMult
- Calculates for every layer of armor

```
// Calculate deflection
var isSharpDmg = def.armorCategory == DamageArmorCategoryDefOf.Sharp;
//var rand = UnityEngine.Random.Range(penAmount - PenetrationRandVariation, penAmount + PenetrationRandVariation);
var deflected = isSharpDmg && armorAmount > penAmount;

// Apply damage reduction
var defCE = def.GetModExtension<DamageDefExtensionCE>() ?? new DamageDefExtensionCE();
var noDmg = deflected && defCE.noDamageOnDeflect;
var newPenAmount = penAmount - armorAmount;

var dmgMult = noDmg ? 0 : penAmount == 0 ? 1 : Mathf.Clamp01(newPenAmount / penAmount);
deflected = deflected || dmgMult == 0;
var newDmgAmount = dmgAmount * dmgMult;
newPenAmount -= partDensity;    // Factor partDensity only after damage calculations
```

```
if (!deflected || !isSharpDmg)
{
    dmgAmount = Mathf.Max(0, newDmgAmount);
    penAmount = Mathf.Max(0, newPenAmount);
}
return !deflected;
```

Deflected Sharpdmg

- Non-blunt, non-partial penetration sharp damage transforms into:
  - DamageType = blunt;
  - localDmgAmount = Mathf.Pow(localPenAmount * 10000, 1 / 3f) / 10;
- $damage = \frac{(\text{bluntPen} \times 10000)^{\frac{1}{3}}}{10}$
- Blunt Penetration keeps the same

Deflected SharpDmg

Sharp	Blunt
Base value: 16 mm RHA Quality multiplier: x80% Multiplier for health 390 / 400: x100% General value: 12.8	Base value: 34 MPa Quality multiplier: x80% Multiplier for health 390 / 400: x100% General value: 27.2

Body part MPa	0.72 MPa
Body part RHA	0.22 mm RHA

- 5.56x45mm FMJ：

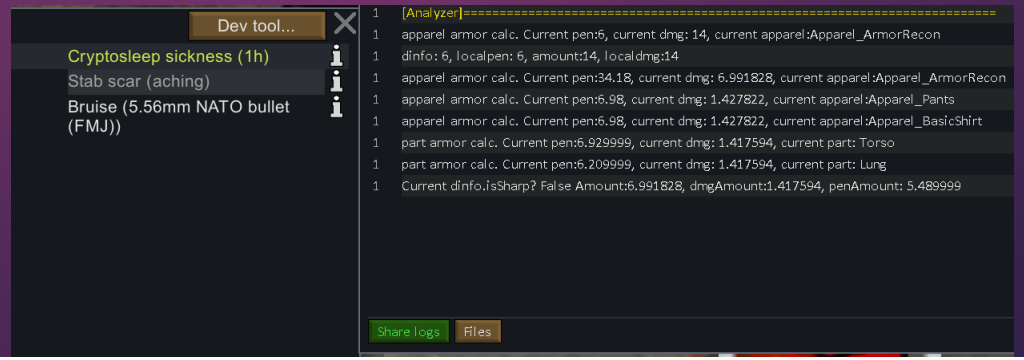
```
5.56mm NATO bullet (FMJ):
  Damage: 14 (Bullet)
  Sharp penetration: 6.00 mm RHA
  Blunt penetration: 34.18 MPa

Final value: 1
```

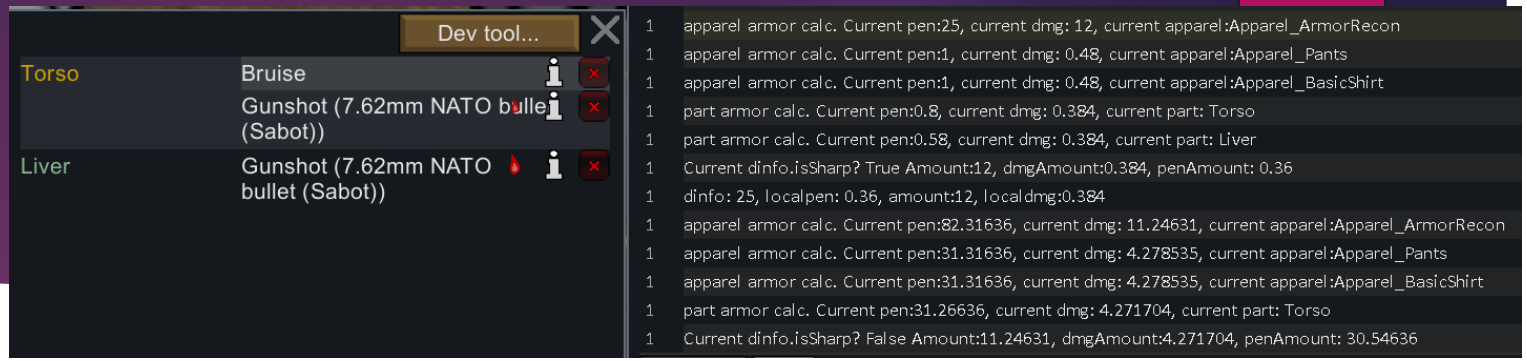
- hitPart：torso -> 12.8 mm RHA > 6.00mm RHA
- Deflected, transform into blunt damage
- damage= $\frac{(34.18 \cdot 10000)^{\frac{1}{3}}}{10} = 6.9918271963046$
- One more calculation

$$b = 6.991828 \cdot \frac{34.18 - 27.2}{34.18} = 1.4278221018139$$

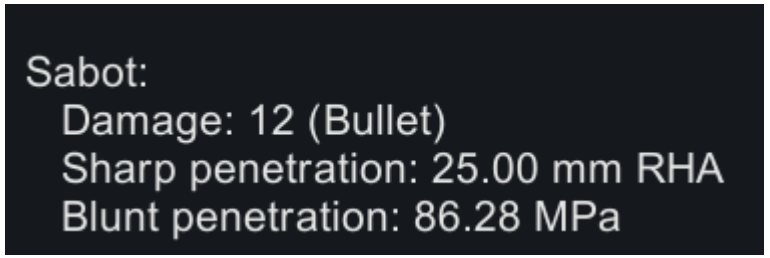
- Torse -> 27.2MPa < 34 Mpa
- Blunt damage never gets deflected, but damage will be reduced by 1- (remain blunt pen/ initial blunt pen), so does its blunt penetration
- At bodypart calculation, since blunt won't hurt inside it only calculates for torso
- Final result : torso receives 1.4 bruise damage

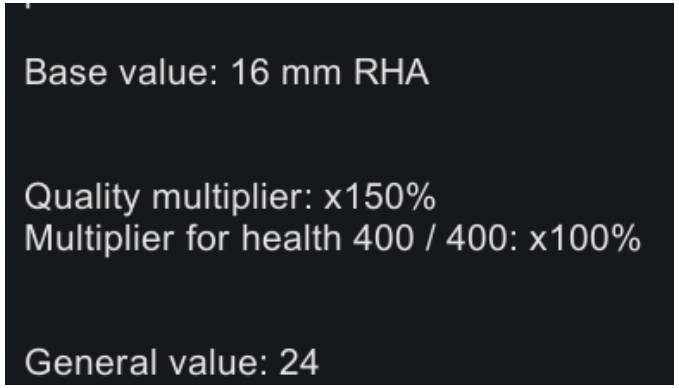


“Partial Pen”



- Sharp damage with its penetration > armor but not big enough to let the damage not getting reduced
  - When (initial pen – armor / initial pen) ≥ 1 the damage multiplier caps at 1, so partial penetration only happens when “initial pen < 2x armor”
  - In this situation, the “partial blunt penetration” is:
    - Initial blunt pen ×((Initial sharp pen – remaining sharp pen) × (Initial dmg – current sharp dmg) / Initial dmg)
  - Damage also adjusts by the formula at p20。





Conclusion

- In CE:
  - You never hit someone with armor rating higher than your armor penetration
  - Having not high enough sharp pen will result in partial penetration
  - Soft armor & hard armor react differently towards damage

Bug(?)

5.56mm NATO bullet (FMJ):
Damage:
5 (EMP)
14 (Bullet)
Sharp penetration: 6.00 mm RHA
Blunt penetration: 34.18 MPa

Final value: 1

	Adult
Armor - Sharp	16.80 ~ 28.00 mm RHA
Armor - Blunt	35.70 ~ 59.50 MPa

- When main damage is environmental (e.g. heat), the secondary damage fully ignores armor

- Reason：Secondary damage is applied in a postfix, where it only checks if the damage is absorbed

```
internal static void Postfix(DamageInfo dinfo, Pawn pawn)
{
    if (shieldAbsorbed)
    {
        return;
    }

    if (dinfo.Weapon?.projectile is ProjectilePropertiesCE props && !props.secondaryDamage.NullOrEmpty() && !_applyingSecondary)
    {
        _applyingSecondary = true;
        foreach (var sec in props.secondaryDamage)
        {
            if (pawn.Dead || !Rand.Chance(sec.chance))
            {
                break;
            }
            var secDinfo = sec.GetDinfo(dinfo);

            pawn.TakeDamage(secDinfo);
        }

        _applyingSecondary = false;
    }
}
```