

M.A.W.S.

Building a walking mech in Arma Reforger – techniques and Enforce Script examples.
Online version: <https://docs.seebie90gaming.com>

- ◆ No copies or uploads of M.A.W.S. assets
- ◆ Example code is free to use
- ◆ For personal reading only – no printed reproduction or modification of this documentation
- ◆ Suggestions: Discord Seebie90 I Nomad – discord.gg/gtg-german-tactical-group

M.A.W.S. WALKER

Building a walking mech in Arma Reforger

Enforce Script basics

Your first walker – step by step

Glossary

Architecture & networking

Leg drive

Procedural leg rig

Hop & thruster jump

Targeting system

Torso & aim stabilization

Loadout & modules

Failure & wreck

View steering & shared turret seats

The mod at a glance

M.A.W.S. License

SCI-FI EXPANSION PACK

Extending a mod without touching it

Energy shield

Gravity field

Building a walking mech in Arma Reforger

How the M.A.W.S. walker moves, aims, swaps modules and seats its crew – explained with short, self-contained Enforce Script examples. Written for modders who want to build something similar.

M.A.W.S. (Mechanized Assault Walker System) is a 14 m, 50 t walking mech for Arma Reforger. Under the hood it is a normal wheeled `Vehicle` – everything that makes it *walk* is script. These pages show the techniques, not the full source: each example is cut down to the idea, uses neutral names and compiles on its own with little glue.

New to Reforger modding? Start here

What is covered

The [Sci-Fi Expansion Pack](#) tab covers the add-on systems: energy shield and gravity field.

Three rules that shaped everything

1. **One physics body.** The walker is a single dynamic body (the vehicle root). Legs, torso and arms are kinematic child entities that only *look* like they carry it. Physics never fights animation.
2. **Server decides, owner moves, everybody animates.** Requests are validated on the server, the machine that owns the physics applies velocities, and every machine runs the leg animation locally from the motion it sees.
3. **Never trust a tick.** Vehicle entities do not always get `EOnFrame` on machines that do not own them. Everything that must run per frame has a fallback tick (see [Architecture](#)).

ABOUT THE EXAMPLES

All examples use Enforce Script as of Arma Reforger 1.x. Names like `MyWalker_...` are placeholders – prefix them with your own tag.

Enforce Script basics

Everything you need to know before the first line of walker code – for people who can program a little, but have not modded Arma Reforger yet.

The tools

- **Arma Reforger Tools** (free on Steam) contain the **Workbench**: world editor, prefab editor, script editor and the compiler.
- Create your own **addon** in the Workbench launcher (*Create new project*). It gets its own folder under `Documents/My Games/ArmaReforgerWorkbench/addons/`.
- Scripts go into `Scripts/Game/<YourFolder>/` inside the addon. Every `.c` file there is compiled automatically – there is no list to register them in.

Enforce Script in two minutes

Enforce Script looks like C# or Java: classes, methods, `if` / `for`, strong types. A few things are different:

THING	WHAT IT MEANS
<code>vector</code>	three floats (x, y, z). y is up , z is forward. <code>v[0]</code> , <code>v[1]</code> , <code>v[2]</code> read the parts.
<code>vector</code> <code>mat[4]</code>	a transform: three axis vectors (right, up, forward) and the position in <code>mat[3]</code> .
<code>ref</code>	strong reference for class members holding objects (<code>ref array<int> m_aList</code>). Without it the object can be deleted.
<code>override</code>	required when you replace a method of the parent class.
<code>modded class</code> <code>X</code>	changes an existing class (also vanilla ones) everywhere, without copying it.
angles	most engine angles are degrees , but some setters want radians – check the method comment.

Components: how code gets into the game

You do not write a `main()`. You write a **component** and add it to a **prefab** (an entity template, for example a vehicle) in the Workbench. A component always comes as a pair of classes:

EXAMPLE CODE - FREE TO USE

Scripts/Game/MyMod/MyMod_SpeedPrinterComponent.c

```
// 1) the "Class" class: holds data shared by all instances; the editor props make it show up in the Workbench
[ComponentEditorProps(category: "MyMod", description: "Prints the speed of the vehicle")]
class MyMod_SpeedPrinterComponentClass : ScriptComponentClass
{
}

// 2) the component itself: one instance per entity
class MyMod_SpeedPrinterComponent : ScriptComponent
{
    // [Attribute] = editable in the prefab editor, value saved in the prefab
    [Attribute("1.0", desc: "Print every n seconds")]
    protected float m_fInterval;

    protected float m_fTimer;

    //! called once when the entity is created
    override void OnPostInit(IEntity owner)
    {
        super.OnPostInit(owner);
        SetEventMask(owner, EntityEvent.FRAME); // without this, EOnFrame is never called
    }

    //! called every frame
    override void EOnFrame(IEntity owner, float timeSlice)
    {
        m_fTimer += timeSlice; // timeSlice = seconds since the last frame
        if (m_fTimer < m_fInterval)
            return;

        m_fTimer = 0;
        Physics phys = owner.GetPhysics();
        if (phys)
            Print("Speed: " + (phys.GetVelocity().Length() * 3.6).ToString() + " km/h");
    }
}
```

Compile (*Script Editor* → *Compile and Reload*), open a vehicle prefab, *Add component* → your component appears under the category you gave it. Drive the vehicle in the world editor (*Play*) and watch the log console.

TIP

The `Print` output goes to the console of the Workbench and into `console.log` in the profile's `logs` folder. Logging is your main debugging tool – use it generously while building.

Multiplayer in plain words

In multiplayer the same entity exists on the server and on every client. Your component runs on **all** of them. Three questions decide where code has to run:

QUESTION	ANSWER IN CODE
Is this the server?	<code>Replication.IsServer()</code> – decides game rules, damage, who may do what.
Does this machine simulate the physics of this entity?	<code>RplComponent.IsOwner()</code> – only the owner should change velocities.
Is this only for looks?	Run it everywhere: animation, effects, sound, HUD.

Two tools move information around:

- `[RplProp()]` on a member variable: the server sets it, calls `Replication.BumpMe()`, and all clients receive the new value. Optionally a callback (`onRplName`) runs on the clients when it arrives.
- `[RplRpc(...)]` on a method: a call that is executed on another machine (client → server, server → owner, or to everybody).

THE ONE PITFALL EVERYBODY HITS

A client can only call a server RPC on something it

OWNS

– in practice its own `SCR_PlayerController`. RPCs sent from a vehicle component disappear without any error message. See [Architecture](#) for the pattern.

Testing in multiplayer

1. First test in the world editor (singleplayer = server and client in one).
2. Then on a real dedicated server with a separate client. Things that work in the editor often break there, because the client is no longer the server and no longer the physics owner.
3. Read both logs: the server's and the client's `console.log`.

Your first walker – step by step

A complete, small walker you can build in an evening: a vehicle that walks at a set speed, turns, and has two legs that swing with its speed. Everything else on this site builds on this skeleton.

1. Vehicle prefab →

2. Drive component →

3. Two legs →

4. Leg component →

5. Test →

6. Multiplayer

Step 1 – Start from a vehicle

A walker in Arma Reforger is a **vehicle** – that gives you seats, damage, replication and physics for free.

1. In the Resource Browser find a simple wheeled vehicle prefab (for example a car or truck).
2. Right click → *Inherit in addon* (creates your own prefab that inherits everything). Name it `MyWalker.et`.
3. Open it. For now keep the car mesh – you will replace it with your walker body later. Seats keep working.

CHECK

You should now be able to place `MyWalker.et` in a world and drive it like the original car.

Step 2 – Take over the driving

This component reads the driver's normal car input (throttle, brake, steering) and writes the velocity itself. The wheels still exist but no longer decide how fast you go.

EXAMPLE CODE - FREE TO USE

Scripts/Game/MyWalker/MyWalker_SimpleDriveComponent.c

```
[ComponentEditorProps(category: "MyWalker", description: "Simple walker drive: sets the velocity directly")]
class MyWalker_SimpleDriveComponentClass : ScriptComponentClass
{
}

class MyWalker_SimpleDriveComponent : ScriptComponent
{
    [Attribute("15", desc: "Top speed forward (km/h)")]
    protected float m_fMaxSpeedKmh;
```

```

[Attribute("5", desc: "Top speed backwards (km/h)")]
protected float m_fMaxReverseKmh;

[Attribute("2", desc: "Acceleration (m/s per second)")]
protected float m_fAccel;

[Attribute("35", desc: "Turn rate at full steering (degrees per second)")]
protected float m_fTurnRate;

protected float m_fSpeed;           // current walking speed (m/s, + = forward)
protected RplComponent m_Rpl;
protected VehicleWheeledSimulation m_Sim;

override void OnPostInit(IEntity owner)
{
    super.OnPostInit(owner);
    m_Rpl = RplComponent.Cast(owner.FindComponent(RplComponent));
    m_Sim = VehicleWheeledSimulation.Cast(owner.FindComponent(VehicleWheeledSimulation));
    SetEventMask(owner, EntityEvent.FRAME);
}

override void EOnFrame(IEntity owner, float timeSlice)
{
    // Only the machine that simulates this vehicle may move it.
    if (m_Rpl && !m_Rpl.IsOwner())
        return;

    Physics phys = owner.GetPhysics();
    if (!phys || !phys.IsDynamic() || !m_Sim)
        return;

    // --- 1. what does the driver want?
    float throttle = m_Sim.GetThrottle();           // 0..1 (W)
    float steering = m_Sim.GetSteering();           // -1..1 (A/D)
    bool reverse = IsReverseGear();

    float target = throttle * m_fMaxSpeedKmh / 3.6; // km/h -> m/s
    if (reverse)
        target = -throttle * m_fMaxReverseKmh / 3.6;
    if (!m_Sim.EngineIsOn())
        target = 0;

    // --- 2. move the current speed toward the target, but not faster than the acceleration
allows
    float step = m_fAccel * timeSlice;
    m_fSpeed = m_fSpeed + Math.Clamp(target - m_fSpeed, -step, step);

    // --- 3. write the velocity: forward speed along the vehicle's nose, keep the vertical
part (gravity)
    vector forward = owner.GetTransformAxis(2);
    forward[1] = 0;
    forward.Normalize();

    vector v = phys.GetVelocity();
    vector newVel = forward * m_fSpeed;
    newVel[1] = v[1];
    phys.SetVelocity(newVel);

    // --- 4. turning: set the rotation speed around the up axis

```

```

        vector w = phys.GetAngularVelocity();
        w[1] = steering * m_fTurnRate * Math.DEG2RAD;
        phys.SetAngularVelocity(w);
    }

    ///! The gearbox numbers gears from reverse up: everything below neutral is reverse.
    protected bool IsReverseGear()
    {
        int reverseGears = m_Sim.GearboxGearsCount() - m_Sim.GearboxForwardGearsCount() - 1;
        return m_Sim.GetGear() < reverseGears;
    }
}

```

Compile, add `MyWalker_SimpleDriveComponent` to `MyWalker.et`, place it, get in, start the engine and drive.

CHECK

The vehicle accelerates evenly to exactly 15 km/h and stops when you let go – no matter what the engine or gearbox would do. If it turns the wrong way, multiply the steering by -1.

PITFALL

If nothing happens, check that `EOnFrame` runs at all (add a `Print`). Forgetting `SetEventMask(owner, EntityEvent.FRAME)` is the most common reason.

Step 3 – Add two legs

1. In the prefab, add two child entities (right click the vehicle in the hierarchy → add child), name them `LegL` and `LegR`.
2. Give each a `MeshObject` component with a simple mesh (a long box is fine for now).
3. Note the position where each leg should hang from the body (the hip). You will type it into the component.

Step 4 – Make the legs swing

This component runs on **every** machine, because it is only for looks. It measures how fast the vehicle moves (from the position change, which works on every machine) and swings the legs forward and back in opposite phase.

EXAMPLE CODE - FREE TO USE

Scripts/Game/MyWalker/MyWalker_SimpleLegsComponent.c

```

[ComponentEditorProps(category: "MyWalker", description: "Swings two child legs with the walking speed")]
class MyWalker_SimpleLegsComponentClass : ScriptComponentClass

```

```

{
}

class MyWalker_SimpleLegsComponent : ScriptComponent
{
    [Attribute("LegL", desc: "Name of the left leg child entity")]
    protected string m_sLeftName;

    [Attribute("LegR", desc: "Name of the right leg child entity")]
    protected string m_sRightName;

    [Attribute("-1.2 2.5 0", desc: "Left hip position relative to the vehicle (m)")]
    protected vector m_vLeftHip;

    [Attribute("1.2 2.5 0", desc: "Right hip position relative to the vehicle (m)")]
    protected vector m_vRightHip;

    [Attribute("2.0", desc: "Distance covered per full step cycle (m)")]
    protected float m_fStride;

    [Attribute("30", desc: "Maximum leg swing (degrees)")]
    protected float m_fSwingDeg;

    protected IEntity m_LegL;
    protected IEntity m_LegR;
    protected vector m_vLastPos;
    protected float m_fPhase;          // 0..1, one full step cycle
    protected float m_fSpeed;          // smoothed, m/s

    override void OnPostInit(IEntity owner)
    {
        super.OnPostInit(owner);
        m_vLastPos = owner.GetOrigin();
        SetEventMask(owner, EntityEvent.FRAME);
    }

    //! Children are attached a moment after creation: look them up lazily.
    protected void FindLegs(IEntity owner)
    {
        IEntity child = owner.GetChildren();
        while (child)
        {
            if (child.GetName() == m_sLeftName)
                m_LegL = child;
            else if (child.GetName() == m_sRightName)
                m_LegR = child;
            child = child.GetSibling();
        }
    }

    override void EOnFrame(IEntity owner, float timeSlice)
    {
        if (timeSlice <= 0)
            return;
        if (!m_LegL || !m_LegR)
            FindLegs(owner);
        if (!m_LegL || !m_LegR)
            return;
    }
}

```

```

// --- speed along the vehicle's nose, measured from the movement (works on every
machine)
vector pos = owner.GetOrigin();
vector delta = pos - m_vLastPos;
m_vLastPos = pos;
float measured = vector.Dot(delta, owner.GetTransformAxis(2)) / timeSlice;
m_fSpeed = Math.Lerp(m_fSpeed, measured, Math.Clamp(timeSlice * 5, 0, 1)); // smooth it

// --- advance the step cycle: faster walking = faster steps
m_fPhase = m_fPhase + (m_fSpeed / m_fStride) * timeSlice;
m_fPhase = m_fPhase - Math.Floor(m_fPhase); // keep it 0..1

// --- swing amount grows with speed, standing still = legs straight
float amount = Math.Clamp(Math.AbsFloat(m_fSpeed) / 2.0, 0, 1);
float swing = Math.Sin(m_fPhase * Math.PI2) * m_fSwingDeg * amount;

PoseLeg(m_LegL, m_vLeftHip, swing);
PoseLeg(m_LegR, m_vRightHip, -swing); // opposite phase
}

//! Rotate a leg around its hip (pitch = forward/back).
protected void PoseLeg(IEntity leg, vector hip, float pitchDeg)
{
    vector mat[4];
    Math3D.AnglesToMatrix(Vector(0, pitchDeg, 0), mat); // yaw, pitch, roll in degrees
    mat[3] = hip;
    leg.SetLocalTransform(mat);
    leg.Update();
}
}

```

CHECK

Standing: legs hang straight. Walking: they swing in opposite directions, faster when you walk faster. If a leg rotates around its middle instead of the hip, the mesh pivot is in the wrong place – move the mesh so its origin is at the top of the leg.

Step 5 – Hide the car

Replace the vehicle's mesh with your body mesh, or hide it, and make the wheels invisible (remove their meshes in the prefab). The wheels keep working as invisible ground contact – that is exactly how M.A.W.S. works.

Step 6 – Test in multiplayer

1. Start a dedicated server and a client with your addon.
2. Drive: the drive component runs only where `IsOwner()` is true, so the walker must still move.

3. Watch another player's walker: its legs must swing too – the leg component measures the movement locally.

KNOWN ISSUE

If the legs of *other* players' walkers do not move on your client, the vehicle's `EOnFrame` is not ticking there. That is an engine behaviour for vehicles you do not own. The fix is a fallback tick, see [Architecture → The fallback tick](#).

Where to go next

YOU WANT

READ

smooth turning toward the camera, no tipping over

[Leg drive](#)

feet that stay on the ground, knees that bend

[Procedural leg rig](#)

a crew that can aim while walking

[Torso & aim stabilization](#)

weapons and parts that can be swapped

[Loadout & modules](#)

START HERE

Glossary

The words used on this site, explained without assuming you already know the engine.

TERM	MEANING
Prefab (<code>.et</code>)	A template for an entity: which components it has and their settings. Placing it in a world creates an entity.
Entity	Anything that exists in the world: a vehicle, a leg part, a tree. Entities can have child entities.
Component	A piece of behaviour attached to an entity. Your scripts are components.
Attribute	A component variable that you can edit in the prefab editor (<code>[Attribute(...)]</code>).
<code>EOnFrame</code>	Method called every rendered frame - after <code>SetEventMask(owner, EntityEvent.FRAME)</code> .
Callqueue	<code>GetGame().GetCallqueue().CallLater(method, ms, repeat)</code> - run a method later or repeatedly.
Server	The machine that runs the game rules. In singleplayer and the world editor your own PC is the server.
Client	A player's game connected to a server.
Owner (physics owner)	The single machine that simulates an entity's physics. Only it should set velocities.
Proxy	A copy of an entity on a machine that is neither server nor owner. It only receives the position.
Replication	Sending state from the server to clients.
<code>RplProp</code>	A replicated member variable. Set on the server, call <code>Replication.BumpMe()</code> .
<code>RplRpc</code>	A method call executed on another machine.
<code>RplId</code>	A network id of a replicated object, used to name an entity in an RPC.
PlayerController	The object that represents a connected player. Clients send their requests through it.
<code>modded class</code>	Changes an existing class everywhere without copying it.

TERM	MEANING
Transform (<code>vector</code> <code>m[4]</code>)	Position and rotation: three axis vectors plus the origin in <code>m[3]</code> .
IK (inverse kinematics)	Computing joint angles so that the end of a chain (the foot) reaches a target.
Gait phase	A number 0..1 that tells where in the step cycle the legs are.
Turret	Engine name for a rotating seat with aim angles - used for gunner, commander and even the walker driver.
Compartment / slot	A seat definition. Turrets and seats hang on slots of the vehicle.

START

Architecture & networking

A walker touches physics, animation, crew input and replication at the same time. Getting the split of responsibilities right is most of the work.

IN PLAIN WORDS

The server makes the decisions, the one machine that simulates the walker's physics moves it, and every machine draws the legs by itself. Players never talk to the walker directly - their requests go through their own player object to the server.

Who computes what

MACHINE	DOES	NEVER DOES
Server	Validates requests, owns replicated state (park, jump counter, module occupancy, marks), runs the AI crew.	Trust a client position or target without checking it.
Physics owner	Writes velocities every frame (<code>RplComponent.IsOwner()</code>). Usually the server, sometimes a driver client.	Decide game state on its own.
Every machine	Animates legs, torso, arms locally from the observed motion; plays effects and sound.	Send per-frame animation data over the network.

Leg poses are **not replicated**. Each machine derives them from how the root moves, so a remote walker costs no bandwidth beyond the vehicle's own transform.

Client requests go through the PlayerController

A client may only send a server RPC through an object it owns. Your vehicle component is not owned by the client, so an RPC sent from it silently disappears – no error, no warning. Route every request through the player's own `SCR_PlayerController` and pass the vehicle's `RplId`.

EXAMPLE CODE - FREE TO USE

Net/MyWalker_PlayerController.c

```

modded class SCR_PlayerController
{
    //! Called on the client by the walker's input component.
    void MyWalker_RequestPark(IEntity walker, bool park)
    {
        RplComponent rpl = RplComponent.Cast(walker.FindComponent(RplComponent));
        if (!rpl)
            return;

        if (Replication.IsServer()) // listen server / singleplayer: no RPC needed
        {
            MyWalker_ServerPark(rpl.Id(), park);
            return;
        }

        Rpc(MyWalker_RpcAsk_Park, rpl.Id(), park);
    }

    [RplRpc(RplChannel.Reliable, RplRcver.Server)]
    protected void MyWalker_RpcAsk_Park(RplId walkerId, bool park)
    {
        MyWalker_ServerPark(walkerId, park);
    }

    protected void MyWalker_ServerPark(RplId walkerId, bool park)
    {
        RplComponent rpl = RplComponent.Cast(Replication.FindItem(walkerId));
        if (!rpl || !rpl.GetEntity())
            return;

        IEntity walker = rpl.GetEntity().GetRootParent();

        // Validate: is the sender really the driver of THIS walker?
        if (!MyWalker_Seats.IsDriverOf(walker, GetControlledEntity()))
            return;

        MyWalker_LegRig rig = MyWalker_LegRig.Cast(walker.FindComponent(MyWalker_LegRig));
        if (rig)
            rig.ServerSetParked(park); // writes an RplProp + Replication.BumpMe()
    }
}

```

CHANNEL CHOICE

Use `RplChannel.Unreliable` for continuous input (driver heading 15×/s) – the next packet comes soon anyway. Use `Reliable` for one-shot actions (park, fire, mark).

Replicated state, local reaction

Events that every machine has to *see* (a jump, a module swap) are replicated as a small state change. Each machine reacts in the `onRplName` callback. Remember that the callback does **not** fire on the machine that wrote the value – the server has to call the reaction itself.

EXAMPLE CODE - FREE TO USE

```
[RplProp(onRplName: "OnJumpReplicated")]
protected int m_iJumpCounter;          // increments per jump: a counter never "misses" an event
protected int m_iJumpSeen;

void ServerStartJump()
{
    m_iJumpCounter++;
    Replication.BumpMe();
    OnJumpReplicated();                // the server does not get its own callback
}

protected void OnJumpReplicated()
{
    if (m_iJumpCounter == m_iJumpSeen)
        return;

    m_iJumpSeen = m_iJumpCounter;
    PlayJumpEffects();                // every machine
    if (IsPhysicsOwner())
        ApplyJumpImpulse();          // only the machine that simulates the body
}
```

The fallback tick

On a dedicated-server client the walker you are *not* driving may never get `EOnFrame`. A watchdog counts real frames; if none arrived for two seconds it re-arms the event mask and, if that does not help, drives all walker components from the callqueue.

EXAMPLE CODE - FREE TO USE

```
protected int m_iRealFrames;
protected bool m_bFallbackRunning;

override void EOnFrame(IEntity owner, float timeSlice)
{
    m_iRealFrames++;
    Tick(owner, timeSlice);
}

//! Every 2 s (CallLater, repeat).
protected void Watchdog()
{
    IEntity owner = GetOwner();
    if (m_iRealFrames > 0)
    {
        m_iRealFrames = 0;
        if (m_bFallbackRunning)    // real tick is back
        {
            GetGame().GetCallqueue().Remove(FallbackTick);
            m_bFallbackRunning = false;
        }
        return;
    }
}
```

```

// step 1: wake the entity
owner.SetFlags(EntityFlags.ACTIVE);
SetEventMask(owner, EntityEvent.FRAME);

// step 2: tick it ourselves
if (!m_bFallbackRunning)
{
    m_bFallbackRunning = true;
    GetGame().GetCallqueue().CallLater(FallbackTick, 0, true);
}
}

protected void FallbackTick()
{
    float dt = GetGame().GetWorld().GetTimeSlice();
    Tick(GetOwner(), dt); // same code path as EOnFrame
}

```

PITFALL

Gate the watchdog to machines that actually render (`!System.IsConsoleApp()`) unless you need the animation on the server too, e.g. because leg parts carry hit-zone colliders.

Leg drive

The walker is a wheeled vehicle with invisible wheels. The leg drive throws away what the engine and gearbox would do and writes the velocity directly – every frame, on the physics owner only.

IN PLAIN WORDS

Instead of letting the engine and wheels push the vehicle, a script sets its speed and turn rate every frame. That gives exact speeds and smooth acceleration. A second part keeps the body from rocking on uneven ground.

Why SetVelocity instead of forces

Wheel forces on a 50 t body react slowly, slide on slopes and depend on gear ratios. Writing the horizontal velocity directly gives an exact top speed, a designed acceleration curve and no sliding when standing. Vertical velocity is left to physics so the body still falls, climbs and lands.

EXAMPLE CODE - FREE TO USE

```
override void EOnFrame(IEntity owner, float dt)
{
    if (m_Rpl && !m_Rpl.IsOwner())
        return; // only the machine that simulates the body

    Physics phys = owner.GetPhysics();
    if (!phys || !phys.IsDynamic() || IsAirborne())
        return; // in the air: hands off (remember the real speed)

    vector fwd = owner.GetTransformAxis(2);
    fwd[1] = 0;
    fwd.Normalize();
    vector right = Vector(fwd[2], 0, -fwd[0]);

    // --- target speed from input, capped by the payload limit
    float vmax = Math.Min(m_fMaxSpeedKmh, GetPayloadSpeedLimitKmh()) / 3.6;
    float target = m_fThrottle * vmax * GetLegFactor(); // 0 when parked or a leg is broken

    // --- ramp: 50 t take a moment to get going
    float rate = m_fDecel;
    if (Math.AbsFloat(target) > Math.AbsFloat(m_fSpeed))
        rate = m_fAccel;
    m_fSpeed += Math.Clamp(target - m_fSpeed, -rate * dt, rate * dt);

    // --- obstacle: never push harder than half of what really happens (no jitter at walls)
    vector v = phys.GetVelocity();
    float real = vector.Dot(v, fwd);
    if (Math.AbsFloat(real) < Math.AbsFloat(m_fSpeed) * 0.5)
```

```

        m_fSpeed = real + (m_fSpeed - real) * 0.5;

        vector vNew = fwd * m_fSpeed + right * m_fStrafe;
        vNew[1] = v[1];           // keep gravity / climbing
        phys.SetVelocity(vNew);

        TurnTowardHeading(owner, phys, dt);
    }

```

Turning toward a heading

The player drives with the mouse: the walker turns toward where the camera looks. A dead zone keeps small glances from turning the body, a gain turns the heading error into a yaw rate. Players can tune both (per-player settings).

EXAMPLE CODE - FREE TO USE

```

protected void TurnTowardHeading(IEntity owner, Physics phys, float dt)
{
    float bodyYaw = owner.GetYawPitchRoll()[0];
    float error = MyMath.WrapDeg(m_fHeadingWanted - bodyYaw);

    float over = Math.Max(Math.AbsFloat(error) - m_fDeadZoneDeg, 0);
    if (error < 0)
        over = -over;

    float maxRate = Math.Lerp(m_fTurnRateSpot, m_fTurnRateWalk,
Math.Clamp(Math.AbsFloat(m_fSpeed) / 2.0, 0, 1));
    float wantDeg = Math.Clamp(over * m_fGain, -maxRate, maxRate);

    // smooth start / stop of the turn
    float k = Math.Clamp(dt / m_fTurnResponse, 0, 1);
    m_fYaw += (wantDeg * Math.DEG2RAD * m_fYawSign - m_fYaw) * k;

    vector w = phys.GetAngularVelocity();
    w[1] = m_fYaw;
    ApplyHullGyro(owner, w, dt);           // pitch / roll, see below
    phys.SetAngularVelocity(w);
}

```

PITFALL

The sign between angular velocity and yaw angle depends on conventions you will get wrong at least once. M.A.W.S. measures it at runtime: if the body turns away from the target for 20 frames, `m_fYawSign` flips.

Hull gyro

With free pitch and roll the body rocks on its invisible suspension over every bump. The gyro pulls pitch/roll rate toward the rate that would align the hull with the *averaged* terrain slope under it. Slopes stay slopes, only the rocking disappears.

EXAMPLE CODE - FREE TO USE

```
protected void ApplyHullGyro(IEntity owner, inout vector w, float dt)
{
    BaseWorld world = owner.GetWorld();
    vector p = owner.GetOrigin();
    float a = 9.0;           // half size of the sampled area (m) - not too small, see below

    float x0 = world.GetSurfaceY(p[0] - a, p[2]);
    float x1 = world.GetSurfaceY(p[0] + a, p[2]);
    float z0 = world.GetSurfaceY(p[0], p[2] - a);
    float z1 = world.GetSurfaceY(p[0], p[2] + a);
    vector ground = Vector(x0 - x1, 2 * a, z0 - z1).Normalized();

    vector up = owner.GetTransformAxis(1);
    // cross(up, ground): rotation axis * sin(angle), horizontal part only
    vector err = Vector(up[1] * ground[2] - up[2] * ground[1], 0, up[0] * ground[1] - up[1] *
ground[0]);
    vector target = err * m_fGyroFollow;           // 1/s

    float k = Math.Clamp(dt / m_fGyroTime, 0, 1); // 0.15 s
    w[0] = w[0] + (target[0] - w[0]) * k;
    w[2] = w[2] + (target[2] - w[2]) * k;
}
```

PITFALL

Do not follow the terrain under the feet too closely. Sampled over ± 5 m and followed directly, the hull pitched and rolled up to $12^\circ/\text{s}$ on uneven ground – every crew camera on it shook. Sample a larger area (± 9 m, two rings averaged) and low-pass the slope (about 0.6 s): the hull follows the shape of the hill, not every bump.

Above about 20° tilt an additional **upright assist** overrides pitch/roll with a stronger righting rate, so a walker that brakes hard while turning on a slope does not tip over.

Park lock

PITFALL

Do not gate the drive on an *animation* value alone. If the rig tick stalls on a server, a parked walker could be driven while every client shows it crouched. Gate on the replicated park wish as well.

EXAMPLE CODE - FREE TO USE

```
bool parked = rig.IsParkWanted() || rig.GetParkBlend() > 0.05;
if (parked)
    m_fThrottle = 0;           // no walking, no strafing, no turning
```

PITFALL

Check in the log that the handbrake really engages. A state flag that starts as "brake already on" means the code never actually pulls it – the invisible wheels roll and a parked walker slowly slides and turns.

Park anchor

Setting the velocity to zero once per frame is not enough. Whatever pushes the body – an editor-placed vehicle that sits a little in the ground and is pushed out again, wheel contacts on a slope – acts in *every* physics substep. A parked walker crept away at about 0.06 m/s and turned 1.4°/s. The fix is an anchor: remember position and heading when parking, then correct the accumulated error every frame on the physics owner.

EXAMPLE CODE - FREE TO USE

```
// physics owner, every frame
if (parked && Math.AbsFloat(m_fSpeed) < 0.1)
{
    vector here = owner.GetOrigin();
    float yaw = MyMath.Yaw(owner);
    float yawErr = MyMath.WrapDeg(m_fAnchorYaw - yaw);

    // first time, or pushed away by a blast / a ramming vehicle: accept the new spot
    if (!m_bAnchor || vector.DistanceXZ(here, m_vAnchor) > 3 || Math.AbsFloat(yawErr) > 20)
    {
        m_bAnchor = true;
        m_vAnchor = here;
        m_fAnchorYaw = yaw;
        yawErr = 0;
    }

    vector back = (m_vAnchor - here) * 2.0;           // 2 m/s per metre of error
    back[1] = 0;
    if (back.Length() > 1)
        back = back.Normalized();

    vector v = physics.GetVelocity();
    physics.SetVelocity(Vector(back[0], v[1], back[2])); // keep gravity

    vector w = physics.GetAngularVelocity();
    w[1] = Math.Clamp(yawErr * 2.0, -10, 10) * Math.DEG2RAD;
    physics.SetAngularVelocity(w);
}
else
{
```

```
m_bAnchor = false;  
}
```

NOTE

Clients that do not own the physics still simulate a dynamic proxy. While parked, zero its horizontal velocity and yaw rate as well – otherwise it drifts between two network corrections.

PITFALL

Re-anchor on a jump within one frame (more than about 0.5 m or 2°). Creep is around a millimetre per frame, a Game Master moving or rotating the walker is not – without this the anchor pulls it back.

Procedural leg rig

The legs are not animated clips. Every machine computes where the feet stand, when they swing and how the joints bend – from nothing more than the motion of the vehicle root.

IN PLAIN WORDS

Each machine looks at how fast the walker moves and decides where each foot should stand. A foot on the ground stays exactly where it is; a lifted foot swings to the next spot. Simple trigonometry then bends the knee so the leg reaches the foot.

Measure, do not replicate

The rig reads speed and yaw rate from how the root moved since the last frame. On machines that only see the interpolated transform, network corrections move a parked walker by centimetres – enough to start a step. Measure over a short window and use small dead zones there.

EXAMPLE CODE - FREE TO USE

```
vector delta = rootPos - m_vLastRootPos;
float measured = Vector(delta[0], 0, delta[2]).Length() / measDt;

if (IsProxy())           // not server, not owner: smooth over 0.2 s, ignore tiny corrections
    measured = SmoothedOverWindow(measured, 0.2);
if (measured < 0.3)
    measured = 0;
```

Gait phase

A single phase value 0..1 runs the whole gait. The right leg is half a cycle behind the left one. Each leg swings for the first 40 % of its cycle – below 50 % there is always a moment with both feet on the ground.

EXAMPLE CODE - FREE TO USE

```
// stride grows with speed, cadence follows from it
float StrideAt(float v)
{
    float s = m_fStrideLength * Math.Pow(Math.Max(v, 0.01) / m_fStrideRefSpeed,
    m_fStrideGrowth);
    return Math.Clamp(s, m_fStrideMin, m_fStrideMax);
}
```

```

void UpdatePhase(float speed, float dt)
{
    float cadence = Math.Clamp(speed / StrideAt(speed), m_fMinCadence, m_fMaxCadence); //
    cycles per second
    m_fPhase = Math.Frac(m_fPhase + cadence * dt);
}

float LegPhase(bool left)
{
    if (left)
        return m_fPhase;
    return Math.Frac(m_fPhase + 0.5);
}

bool IsSwinging(bool left)
{
    return LegPhase(left) < m_fSwingFraction; // 0.4
}

```

Stance and swing

- **Stance:** the foot is fixed in *world* space. It stays exactly where it landed while the body walks over it.
- **Swing target** (Raibert style): the neutral foot spot, moved ahead by $\text{velocity} \times \tau$ and rotated by $\text{yaw rate} \times \tau$, where τ = remaining swing time + half the stance. Then traced to the ground.
- **Swing arc:** $4s(1-s) \cdot \text{stepHeight}$ on top of the eased line between lift-off and target, never below the terrain.
- **Rescue step:** if a planted foot would overstretch the leg, it steps out of phase.

EXAMPLE CODE - FREE TO USE

```

//! Ground under a foot target. Own walker parts are skipped by retrying just below the hit.
vector TraceGround(BaseWorld world, IEntity owner, vector p)
{
    vector from = p + Vector(0, 5, 0);
    vector to = p - Vector(0, 10, 0);
    for (int i = 0; i < 3; i++)
    {
        auto ptr TraceParam tp = new TraceParam();
        tp.Start = from;
        tp.End = to;
        tp.Flags = TraceFlags.ENTS | TraceFlags.WORLD;
        tp.LayerMask = EPhysicsLayerDefs.Projectile;
        tp.Exclude = owner;

        float f = world.TraceMove(tp, null);
        if (f >= 1.0)
            break;

        vector hit = from + (to - from) * f;
        if (tp.TraceEnt && tp.TraceEnt.GetRootParent() == owner)
        {

```

```

        from = hit - Vector(0, 0.05, 0);    // our own leg: look below it
        continue;
    }
    return hit;
}
return Vector(p[0], world.GetSurfaceY(p[0], p[2]), p[2]);
}

```

Three segments, solved as two

The walker leg has thigh, shin and metatarsus (digitigrade, like a bird). Keeping thigh and metatarsus parallel turns it into a pantograph – and the problem collapses to a classic two-bone solve with the law of cosines.

EXAMPLE CODE - FREE TO USE

```

//! Two-bone IK in the leg plane. Returns the knee bend angle (rad) for a hip-to-ankle distance d.
//! a, b = effective segment lengths (pantograph reduced).
static float KneeAngle(float a, float b, float d)
{
    d = Math.Clamp(d, Math.AbsFloat(a - b) + 0.001, a + b - 0.001);
    float c = (a * a + b * b - d * d) / (2.0 * a * b);
    return Math.Acos(Math.Clamp(c, -1.0, 1.0));
}

//! Angle between the hip-to-ankle line and the first segment.
static float HipOffset(float a, float b, float d)
{
    float c = (a * a + d * d - b * b) / (2.0 * a * d);
    return Math.Acos(Math.Clamp(c, -1.0, 1.0));
}

```

Moving kinematic parts

Each leg segment is its own child entity. The rig computes a target transform in the **root frame**, converts it into the frame of the real parent and writes it.

EXAMPLE CODE - FREE TO USE

```

void WritePart IEntity part, IEntity parent, vector targetInRoot[4], vector parentInRoot[4])
{
    vector local[4];
    Math3D.MatrixInvMultiply4(parentInRoot, targetInRoot, local);
    part.SetLocalTransform(local);
    part.Update();
}

```

PITFALL

Never read `GetLocalTransform` of freshly attached children – it returns garbage in the first frame. Collect the parts with a short `CallLater` after spawn and compute their rest pose from your own data.

Pelvis

The pelvis carries the life of the walk: bob (twice per cycle), sway and roll (once per cycle), forward lean from speed and acceleration, a small height adaption when a planted leg would overstretch, and a spring for impacts. The `stabilizer` later removes exactly the fast part of this from the torso.

EXAMPLE CODE - FREE TO USE

```
float ang = m_fPhase * Math.PI2;
float bob  = m_fBobAmp  * m_fGaitAmp * Math.Cos(2 * ang);
float sway = m_fSwayAmp * m_fGaitAmp * Math.Cos(ang);
float roll = m_fRollAmp * m_fGaitAmp * Math.Cos(ang);

pelvis = posture(parkBlend) * level(slope) * Euler(twist, lean, roll);
pelvis[3] = pelvis[3] + Vector(sway, bob - m_fHeightAdapt - limp, 0) + m_vImpactSpring;
```

NOTE

Parking blends a crouch posture in (`SmoothCD` toward 1). Limp lowering is faded out while crouched – otherwise a walker with two broken legs folds its legs sideways.

MOVEMENT

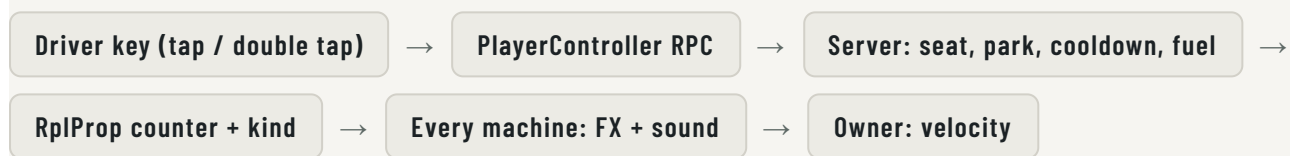
Hop & thruster jump

One key, two jumps: a tap makes the legs push off, a double tap fires the thrusters. The server validates, every machine shows flames and sound, and only the physics owner changes the velocity.

IN PLAIN WORDS

Pressing jump asks the server. The server checks fuel and cooldown, then counts up a number that every machine receives. Every machine shows flames and plays sound; only the machine that moves the walker pushes it into the air.

Flow



Telling tap and double tap apart

The first press sends a hop immediately. A second press within 0.45 s upgrades it to a thruster jump – the hop has already started its dip, so there is no input lag.

EXAMPLE CODE - FREE TO USE

```
protected float m_fLastPress = -10;

protected void OnJumpKey(float value, EActionTrigger reason)
{
    float now = GetGame().GetWorld().GetWorldTime() * 0.001;
    int kind = KIND_HOP;
    if (now - m_fLastPress < m_fDoubleTapTime)
        kind = KIND_JET;
    m_fLastPress = now;

    SCR_PlayerController pc = SCR_PlayerController.Cast(GetGame().GetPlayerController());
    if (pc)
        pc.MyWalker_RequestJump(GetOwner().GetRootParent(), kind);
}
```

Server: validate, then count

EXAMPLE CODE - FREE TO USE

```
[RplProp(onRplName: "OnJumpReplicated")]
protected int m_iJumpNr;
[RplProp(onRplName: "OnJumpReplicated")]
protected int m_iJumpKind;
[RplProp()]
protected float m_fFuel = 100;

void ServerJump(int kind)
{
    if (IsParked() || OnCooldown())
        return;
    if (kind == KIND_JET && m_fFuel < m_fFuelPerJump)
        kind = KIND_HOP;           // not enough fuel: still hop

    if (kind == KIND_JET)
        m_fFuel -= m_fFuelPerJump; // refuels over time (Calllater)

    m_iJumpKind = kind;
    m_iJumpNr++;
    Replication.BumpMe();
    OnJumpReplicated();
}
```

Every machine: effects. Owner: physics.

EXAMPLE CODE - FREE TO USE

```
protected void OnJumpReplicated()
{
    if (m_iJumpNr == m_iJumpSeen)
        return;
    m_iJumpSeen = m_iJumpNr;

    if (m_iJumpKind == KIND_JET)
    {
        m_fBurn = m_fBurnTime;           // 1.6 s, the per-frame tick shows flames while > 0
        return;
    }

    // hop: dip first, stretch and lift off a moment later
    m_Rig.AddBodyImpulse(Vector(0, -m_fHopDip, 0));
    GetGame().GetCallqueue().CallLater(HopStretch, m_fHopDipTime * 1000, false);
}

//! Per frame on the owner while burning.
protected void BurnTick(Physics phys, float dt)
{
    vector v = phys.GetVelocity();
    v = v + Vector(0, m_fJetUp * dt, 0) + m_vForward * (m_fJetForward * dt);
}
```

```
    phys.SetVelocity(v);  
}
```

Braking burn and landing

- **Braking burn:** below 14 m above ground and sinking faster than 7 m/s the nozzles fire again (20 m/s² up) until the sink rate is down to 3 m/s.
- **Landing damper:** close to the ground the sink rate is clamped to 3.5 m/s – the leg rig absorbs the rest visually. Hard landings otherwise pushed the hidden wheels through the terrain.
- **Ground rescue** (always active): if the root ends up more than 0.6 m below the terrain surface, it is lifted out.

PITFALL: SOUND

Sound sources are not unlimited. The hop sound and the thruster ignition 0.2 s later were both played through the same `SoundComponent` on the pelvis – the ignition was swallowed. One-shots that overlap with other sounds are now played free via `AudioSystem.PlayEvent(project, event, transform)`.

EXAMPLE CODE - FREE TO USE

```
AudioHandle PlayFree(ResourceName project, string eventName, vector pos)  
{  
    vector m[4];  
    Math3D.MatrixIdentity4(m);  
    m[3] = pos;  
    return AudioSystem.PlayEvent(project, eventName, m);  
}
```

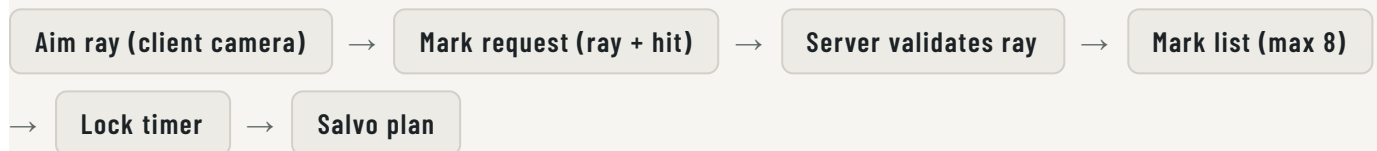
Targeting system

The commander's shoulder launchers never fire blind. A target is set first – a locked vehicle, a locked ground point or a map assignment – and the server checks every step.

IN PLAIN WORDS

The commander points at something and presses the mark key. The server checks the request, adds the target to a list and waits until it is locked. Rockets only fire at targets from that list.

Flow



The aim ray

The ray starts at the camera and skips the own walker: torso, arms and launchers are in the way of almost every view, so every own hit moves the start just past it and traces again.

EXAMPLE CODE - FREE TO USE

```

bool AimRay(IEntity walker, IEntity localChar, out vector hit, out IEntity hitVehicle, out bool
hasHit)
{
    vector cam[4];
    GetGame().GetWorld().GetCurrentCamera(cam);
    vector dir = cam[2];
    vector from = cam[3];
    vector to = from + dir * m_fRange;          // 2 km, more with a laser module

    array<IEntity> skip = {walker, localChar};
    for (int i = 0; i < 5; i++)
    {
        autoPtr TraceParam tp = new TraceParam();
        tp.Start = from;
        tp.End = to;
        tp.Flags = TraceFlags.ENTS | TraceFlags.WORLD | TraceFlags.OCEAN;
        tp.LayerMask = EPhysicsLayerDefs.Projectile;
        tp.ExcludeArray = skip;

        float f = GetGame().GetWorld().TraceMove(tp, null);
    }
}
  
```

```

    if (f >= 1.0)
        break;

    hit = from + (to - from) * f;
    IEntity e = tp.TraceEnt;
    if (e && e.GetRootParent() == walker)
    {
        from = hit + dir * 0.3;    // own part: continue behind it
        continue;
    }

    hasHit = true;
    if (e)
        hitVehicle = Vehicle.Cast(e.GetRootParent());
    return true;
}

hit = to;
hasHit = false;
return true;
}

```

Server-validated marks

The client sends ray start, direction, hit point and the hit vehicle's `RplId`. The server does not trust the hit point: it must lie close to the ray (5 m + 1 % of the distance), and the ray must start near the walker.

EXAMPLE CODE - FREE TO USE

```

bool PointOnRay(vector origin, vector dir, vector point)
{
    vector toPoint = point - origin;
    float along = vector.Dot(toPoint, dir);
    if (along < 0)
        return false;
    float off = (toPoint - dir * along).Length();
    return off <= 5.0 + 0.01 * along;
}

```

Then the best candidate wins, in this order: the vehicle the ray hit → the best sensor track in a narrow cone → the "next target" cursor → a ground point (needs terrain line of sight, no friendlies close by). Marking the same thing again removes it.

Locks

A marked track is not fireable immediately. Lock time grows with distance (1 s at 500 m to 8 s at 5 km), needs line of sight (unless a datalink module is fitted) and decays after 1.5 s out of sight.

EXAMPLE CODE - FREE TO USE

```
float LockTime(float dist)
{
    float t = Math.InverseLerp(500, 5000, dist);
    return Math.Lerp(1.0, 8.0, Math.Clamp(t, 0, 1));
}
```

Salvo plan

A salvo walks round-robin over all fireable marks, so four rockets at four targets hit each one once. Marks can be assigned to specific weapons from the map (a bitmask per mark); a salvo of weapon X skips marks assigned elsewhere.

EXAMPLE CODE - FREE TO USE

```
int Rotate(array<ref MySalvoTarget> plan)
{
    int start = m_iPlanOffset % plan.Count();
    for (int i = 0; i < start; i++)
    {
        plan.Insert(plan[0]);
        plan.RemoveOrdered(0);
    }
    m_iPlanOffset += m_iRocketsThisSalvo;
    return start;
}
```

PITFALL

If there are marks but none of them is fireable, refuse the salvo. Silently falling back to "fire at the crosshair" once sent rockets into the sky.

The commander lock cross

Since players fired without a target, the commander now sees a small cross in the screen centre. Below it the HUD says what the mark key would lock there – LOCK VEHICLE , LOCK POINT TRACK or NO TARGET – followed by the key the player has bound. The binding is read at runtime, so rebinding is reflected immediately.

EXAMPLE CODE - FREE TO USE

```
string BoundKey(string action)
{
    InputManager im = GetGame().GetInputManager();
    EInputDeviceType dev = EInputDeviceType.KEYBOARD;
    if (im.GetLastUsedInputDevice() == EInputDeviceType.GAMEPAD)
        dev = EInputDeviceType.GAMEPAD;
```

```
InputBinding binding = im.CreateUserBinding();
array<string> list = {};
if (!binding.GetBindings(action, list, dev) || list.IsEmpty())
    return string.Empty;

return PrettyKey(list[0]);    // "keyboard:KC_I" -> "I",
"gamepad0:shoulder_left+gamepad0:a" -> "LB + A"
}
```

Torso & aim stabilization

The legs bounce, the torso must not. And the gunner's sight should stay on the target while the walker walks and turns – without ever fighting the mouse.

IN PLAIN WORDS

The upper body ignores the bouncing of the hips. The gunner's sight remembers the point it was aimed at and keeps pointing there while the walker moves - until the player moves the mouse.

Torso: keep the slow part, drop the fast part

The torso hangs on the pelvis, but only on its *posture* (crouch, slope levelling, slow settling). Bob, sway, roll and lean are taken out. This runs in the same frame as the leg rig, right after the pelvis – a frame of lag would show as jitter.

EXAMPLE CODE - FREE TO USE

```
void ApplyUpperBody(MyLegRig rig, MyPartRegistry reg, float dt)
{
    vector posture[4];
    rig.GetPelvisPostureLocal(posture);    // crouch + slope, no bob/sway/roll

    vector settle[4];
    rig.GetPelvisSettleLocal(settle);    // slow height adaption: follow it, or the waist opens a
    gap
    vector torsoBase[4];
    Math3D.MatrixMultiply4(posture, settle, torsoBase);

    vector yaw[4];
    Math3D.AnglesToMatrix(Vector(m_fTorsoYaw, 0, 0), yaw);
    yaw[3] = m_vTorsoPivot - rig.GetPelvisRestOrigin();

    vector torso[4];
    Math3D.MatrixMultiply4(torsoBase, yaw, torso);
    reg.SetTarget(PART_TORSO, torso);
    reg.Write(PART_TORSO);
}
```

Fast arms, slow torso

A 20 t torso that snaps to the mouse looks wrong. The arms take small and fast corrections inside a $\pm 35^\circ$ window; the torso follows lazily (gain $\times$ remaining angle, max $80^\circ/\text{s}$, 3° dead band).

EXAMPLE CODE - FREE TO USE

```
float FollowTorso(float aimYaw, float dt)
{
    float err = MyMath.WrapDeg(aimYaw - m_fTorsoYaw);
    float over = Math.Max(Math.AbsFloat(err) - m_fTorsoDeadband, 0);
    float speed = Math.Min(m_fTorsoMaxSpeed, m_fTorsoGain * over);
    m_fTorsoYaw += Math.Clamp(err, -speed * dt, speed * dt);

    // never let the aim leave the arm window
    m_fTorsoYaw = aimYaw - Math.Clamp(MyMath.WrapDeg(aimYaw - m_fTorsoYaw), -m_fArmWindow, m_fArmWindow);
    return m_fTorsoYaw;
}
```

Convergence on the measured aim point

The two arm weapons sit 6 m apart. Instead of a fixed convergence distance, a ray from the gunner's eye along the sight measures the real distance (every 0.1 s, smoothed in log space so 20→40 m feels like 500→1000 m) and each barrel line is aimed exactly through that point.

The aim point lock

The engine has no turret stabilization API, and simply re-applying the old direction every frame overwrote the player's input – the turret could not be turned while walking. The working solution locks a **world point**:

1. While the gunner gives aim input, do nothing and keep re-taking the point under the crosshair.
2. When input stops, the last point stays locked.
3. Every frame without input, compute the angles to that point from the current turret base and set them.

EXAMPLE CODE - FREE TO USE

```
bool PlayerIsAiming(IEntity localChar)
{
    ChimeraCharacter ch = ChimeraCharacter.Cast(localChar);
    if (!ch || !ch.GetCharacterController())
        return false;
    CharacterInputContext ctx = ch.GetCharacterController().GetInputContext();
    return ctx && ctx.GetAimChange().LengthSq() > 1e-10;
}

void HoldPoint(TurretComponent turret, TurretControllerComponent ctrl, vector baseMat[4], bool aiming)
{
    vector dir = turret.GetAimingDirectionWorld().Normalized();
    vector eye = EyePosition(dir);

    if (aiming || !m_bHasPoint)
```

```

{
    m_vPoint = eye + dir * TraceDistance(eye, dir); // re-take the point
    m_bHasPoint = true;
    return;
}

vector local = MyMath.DirToLocal(baseMat, m_vPoint - eye).Normalized();
vector m[4];
Math3D.DirectionAndUpMatrix(local, "0 1 0", m);
vector want = Math3D.MatrixToAngles(m); // degrees: yaw, pitch

vector cur = turret.GetAimingRotation(); // degrees
float dYaw = MyMath.WrapDeg(want[0] - cur[0]);
float dPitch = want[1] - cur[1];
if (Math.AbsFloat(dYaw) + Math.AbsFloat(dPitch) < 0.02)
    return;

ctrl.SetAimingAngles((cur[0] + dYaw) * Math.DEG2RAD, (cur[1] + dPitch) * Math.DEG2RAD); //
RADIANS
}

```

PITFALL

`SetAimingAngles` takes

RADIANS

, `GetAimingRotation` returns

DEGREES

. Passing degrees drives both axes into their limits and freezes them there.

PITFALL

Do not measure small rotations with `acos(dot)` : near 1 it produces $\sim 0.02^\circ$ of noise per frame, which looks like constant motion. Use the length of the cross product (= sin of the angle).

NOTE

Locking a point instead of a direction also compensates the walker's own travel: at 8 m/s a target at 300 m drifts $1.5^\circ/\text{s}$ out of a direction-held sight.

PITFALL

For a turret whose direction *is* the walking heading (the driver), do not lock the point under the crosshair. The driver usually looks slightly down, so that point lies on the ground a few metres ahead – walking towards and past it turns the walker by itself. Lock a point far along the sight instead (5 km, no trace).

PITFALL

Do not fall back to re-applying the old direction every frame either: it overwrites the player's mouse input and the walker cannot be turned at all. Keep the input priority of the point lock.

PITFALL

Re-take the point when it gets close (under about 25 m) or when a single frame would need more than 3° of correction. Walking past a locked point on the ground next to the walker otherwise swings the turret around at hundreds of degrees per second.

PITFALL

Detect teleports. If the turret base jumps by more than about 10° or 5 m in a single frame (Game Master moved the walker), treat it like aim input and re-take the point – otherwise the walker turns back to its old heading.

Loadout & modules

Five hardpoints, dozens of modules, swappable in the field. The whole loadout replicates as a single integer – every machine builds the module meshes itself.

IN PLAIN WORDS

Which weapon sits on which mount is stored as one number on the server. Every machine reads that number and attaches the matching weapon models itself.

Hardpoints and module definitions

HARDPOINT	CLASS	HANGS ON
HP_Arm_L / HP_Arm_R	ARM	arm parts
HP_Shoulder_L / HP_Shoulder_R	SHOULDER	launcher parts
HP_Back	BACK	torso

A module fits only a hardpoint of the same class. Each definition carries an id, label, class, weight in tonnes, swap time and the prefabs for the left and right side.

One int for the whole loadout

EXAMPLE CODE - FREE TO USE

```
static const int BITS = 5;          // up to 31 modules per hardpoint, 0 = empty

[RplProp(onRplName: "OnLoadoutReplicated")]
protected int m_iLoadout;

int GetModuleAt(int hp)
{
    return (m_iLoadout >> (hp * BITS)) & 31;
}

void ServerSetModule(int hp, int module)
{
    int mask = 31 << (hp * BITS);
    m_iLoadout = (m_iLoadout & ~mask) | (module << (hp * BITS));
    Replication.BumpMe();
    Rebuild();                      // onRplName does not fire on the server itself
}
```

```
protected void OnLoadoutReplicated()
{
    Rebuild();
}
```

Build locally on every machine

Module entities have no `RplComponent`. Each machine compares the replicated loadout with what it has mounted and spawns or deletes the difference itself. That keeps network entity count low and makes late joiners correct automatically.

EXAMPLE CODE - FREE TO USE

```
protected void Rebuild()
{
    for (int hp = 0; hp < HardpointCount(); hp++)
    {
        int want = GetModuleAt(hp);
        if (m_aMounted[hp] == want)
            continue;

        if (m_aEntities[hp])
            SCR_EntityHelper.DeleteEntityAndChildren(m_aEntities[hp]);
        m_aEntities[hp] = null;
        m_aMounted[hp] = want;
        if (want == 0)
            continue;

        IEntity part = GetPartFor(hp);
        EntitySpawnParams sp = new EntitySpawnParams();
        sp.TransformMode = ETransformMode.WORLD;
        part.GetWorldTransform(sp.Transform);

        IEntity e = GetGame().SpawnEntityPrefab(Resource.Load(PrefabFor(hp, want)),
part.GetWorld(), sp);
        part.AddChild(e, -1);
        e.SetLocalTransform(OffsetFor(hp));
        m_aEntities[hp] = e;

        // new children report garbage transforms in their first frame: re-scan weapons a bit
        later
        GetGame().GetCallqueue().CallLater(RescanWeapons, 400, false, hp);
    }
}
```

Weight and speed

Payload is the sum of the module weights. Top speed = 60 – 1 km/h per tonne (minimum 5), ×0.75 when overloaded above 20 t. The leg drive uses it as `vmax`, and a limiter on the physics owner clamps the

horizontal velocity in case anything else (a jump, a push) exceeds it.

PITFALL

If your drive has its own top-speed attribute, make sure it is not lower than the payload formula's maximum. Ours was 40 km/h – the payload limit then only mattered above 20 t, and an empty walker never got faster.

Swapping in the field

The commander requests a swap through the PlayerController. The server checks the seat, that the walker is parked and that a depot is near, then runs a swap timer. Each machine shows a progress bar from a small replicated start counter per hardpoint, so the bar restarts on every machine without sending times.

Failure & wreck

At 0 % hull the walker does not just turn into a wreck. One side gives way, it collapses, burns and throws sparks – and seven seconds later it goes up in a huge explosion.

IN PLAIN WORDS

When the hull reaches zero, the server sets one replicated number. Every machine then plays the same collapse and effects by itself. After seven seconds the server sets off the explosion and tells every nearby player to show it.

Flow



One replicated state, local show

Only two small values travel over the network: a state (0 intact, 1 collapsing, 2 exploded) and a random seed that picks the side that gives way first. Every machine plays the rest from these. A player who joins after the explosion gets state 2 directly and sees the wreck lying – without replaying the sequence.

EXAMPLE CODE - FREE TO USE

```

[RplProp(onRplName: "OnWreckRpl")]
protected int m_iWreck;           // 0 intact, 1 collapsing, 2 exploded
[RplProp(onRplName: "OnWreckRpl")]
protected int m_iWreckSeed;
protected int m_iWreckSeen;

void ServerStartWreck()
{
    if (m_iWreck != 0)
        return;

    m_iWreck = 1;
    m_iWreckSeed = Math.RandomInt(1, 1000000);
    Replication.BumpMe();
    OnWreckRpl();                  // the server gets no callback for its own change
    GetGame().GetCallqueue().CallLater(ServerExplode, 7000, false);
}
  
```

```
protected void OnWreckRpl()
{
    if (m_iWreck == m_iWreckSeen)
        return;

    int before = m_iWreckSeen;
    m_iWreckSeen = m_iWreck;

    bool instant = before == 0 && m_iWreck == 2;    // joined after the explosion
    m_Rig.SetWreck(true, m_iWreckSeed, instant);
    if (!instant && before == 0 && !System.IsConsoleApp())
        StartEffectTimeline();                    // sparks, smoke, impact sound, fire
}
```

NOTE

A Game Master "Destroy" sets the vanilla hit zone straight to 0. Watch for the DESTROYED state as well and play the sequence anyway.

The collapse is a pose

No animation clip: the collapse is laid on top of the existing park crouch. First a stagger (one side gives way, a short knee jerk), then a fall that accelerates like a free fall into a deeper, nose-down, rolled pose, ending with a kick into the impact spring. The stabilizer adds a forward slump to the torso and lets the arms hang.

EXAMPLE CODE - FREE TO USE

```
m_fWreckT += dt;
float stagger = Math.Clamp(m_fWreckT / 1.0, 0, 1);
stagger = stagger * stagger * (3 - 2 * stagger);    // smoothstep
float u = Math.Clamp((m_fWreckT - 1.0) / 1.1, 0, 1);
float fall = u * u;                                // accelerating like a free fall

m_fParkBlend = Math.Max(m_fParkBlendAtStart, 0.25 * stagger + 0.75 * fall);
m_fWreckBlend = fall;
m_fWreckRoll = 0.45 * stagger + 0.55 * fall;

// posture = park crouch + wreck: deeper, nose down, rolled to the side that gave way
float pitch = m_fParkPitch * m_fParkBlend + 14 * m_fWreckBlend;
float roll = m_fWreckSide * 11 * m_fWreckRoll;
vector offset = m_vParkOffset * m_fParkBlend + Vector(0, -1.4, 0.6) * m_fWreckBlend;
```

NOTE

Because the wreck counts as a park wish, the existing park lock already blocks walking and firing – no extra checks in the drive or the weapons.

The big explosion has to reach everyone

PITFALL

A warhead that the server spawns and triggers explodes *on the server*. In the first test the clients saw the collapse but no explosion at all.

The fix is the same path the missiles use: the server sends effect and sound to every player near the walker through that player's own PlayerController (players with the Game Master editor open get it too). The warhead still does the damage on the server.

EXAMPLE CODE - FREE TO USE

```
// server
vector centre = TorsoCentre();
MyWalker_Blast(WARHEAD, centre, lastAttacker);      // damage, server only
foreach (SCR_PlayerController pc : PlayersNear(centre, 2000))
    pc.MyWalker_SendDetonation(centre, BIG_EFFECT, BIG_SOUND, 4.0); // picture and sound, every client
```

The crew

Whoever is still aboard dies with the explosion and is thrown out. The seats sit on child turret entities, so walk the children. Remember the last attacker of the hull and use it as instigator – that keeps the kill credit.

EXAMPLE CODE - FREE TO USE

```
protected void KillCrew(IEntity ent, Instigator who, int depth)
{
    if (!ent || depth > 6)
        return;

    SCR_BaseCompartmentManagerComponent seats =
    SCR_BaseCompartmentManagerComponent.Cast(ent.FindComponent(SCR_BaseCompartmentManagerComponent))
    ;
    if (seats)
        seats.KillOccupants(who, true);    // eject the bodies

    IEntity child = ent.GetChildren();
    while (child)
    {
        KillCrew(child, who, depth + 1);
        child = child.GetSibling();
    }
}
```

View steering & shared turret seats

The driver steers with the mouse like on foot, the gunner and commander ride on a torso that turns and crouches – and nobody stands up from their seat while it moves.

IN PLAIN WORDS

The driver sits in a turret and walks where they look. Gunner and commander sit in turrets that ride along with the turning, crouching upper body. Personal steering settings are saved in the player profile and sent with the driver's input.

The driver sits in a turret

A vanilla pilot seat routes input through the car simulation. The walker driver instead sits in a **weaponless turret** on the root. Its aim direction *is* the walking heading: look somewhere and the walker turns there. Forward/back and side steps come from a dedicated input context that also swallows the turret turn keys.

EXAMPLE CODE - FREE TO USE

```

//! Driver machine, per frame while seated in the driver turret.
protected void DriverTick(IEntity walker, TurretComponent turret, float dt)
{
    InputManager im = GetGame().GetInputManager();
    im.ActivateContext("MyWalker_DriverContext");
    float fwd = im.GetActionValue("MyWalker_DriveMove");
    float side = im.GetActionValue("MyWalker_DriveStrafe");

    vector d = turret.GetAimingDirectionWorld();
    float heading = Math.Atan2(d[0], d[2]) * Math.RAD2DEG;

    m_fSendTimer -= dt;
    bool changed = Math.AbsFloat(MyMath.WrapDeg(heading - m_fSentHeading)) > 0.5;
    if (m_fSendTimer > 0 && !changed)
        return;

    m_fSendTimer = 0.066;           // ~15 per second, unreliable
    m_fSentHeading = heading;
    SCR_PlayerController pc = SCR_PlayerController.Cast(GetGame().GetPlayerController());
    pc.MyWalker_SendDriverInput(walker, heading, fwd, side, MyDriveSettings.Pack());
}

```

The server checks that the sender really sits in this walker's driver turret, writes heading and inputs into RplProps and the physics owner drives with them (**Leg drive**). The local driver uses its own copy to avoid a round-trip delay.

Per-player settings

Dead zone and sensitivity are personal. They live in a `ModuleGameSettings` class (saved in the player profile like the vanilla settings) and travel with the driver input, so every driver uses their own values on any server.

EXAMPLE CODE - FREE TO USE

```
class MyDriveSettings : ModuleGameSettings
{
    [Attribute("5")]    float m_fHeadingDeadZone;    // deg
    [Attribute("1")]    float m_fSteerSensitivity;    // x gain
    [Attribute("0.15")] float m_fStickDeadZone;      // 0..1

    static float Get(string name, float fallback)
    {
        BaseContainer c = GetGame().GetGameUserSettings().GetModule("MyDriveSettings");
        float v;
        if (c && c.Get(name, v))
            return v;
        return fallback;
    }

    static void Set(string name, float v)
    {
        BaseContainer c = GetGame().GetGameUserSettings().GetModule("MyDriveSettings");
        c.Set(name, v);
        GetGame().UserSettingsChanged();
        GetGame().SaveUserSettings();
    }
}
```

The sliders appear under *Settings* → *Controls* in the mod's key binding category. The keybind sub-menu (`SCR_KeybindSetting`) builds its rows from script, so a modded `ListActionsFromCurrentCategory()` can append vanilla slider widgets after the bindings.

EXAMPLE CODE - FREE TO USE

```
modded class SCR_KeybindSetting
{
    override protected void ListActionsFromCurrentCategory()
    {
        super.ListActionsFromCurrentCategory();

        SCR_KeyBindingCategory cat =
            m_KeybindConfig.m_KeyBindingCategories.Get(m_CategoriesSpinBox.GetCurrentIndex());
        if (!cat || cat.m_sName != "MY_WALKER")
            return;

        Widget row = GetGame().GetWorkspace().CreateWidgets("
{4A41296C0E9A889F}UI/layouts/WidgetLibrary/WLib_Slider.layout", m_wActionsLayout);
        m_aRowWidgets.Insert(row);    // cleared with the other rows on category change
        SCR_SliderComponent s = SCR_SliderComponent.Cast(row.FindHandler(SCR_SliderComponent));
        s.SetLabel("Steering dead zone (mouse)");
        s.SetSliderSettings(0, 20, 0.5, "%1°");
    }
}
```

```

        s.SetValue(MyDriveSettings.Get("m_fHeadingDeadZone", 5));
        s.GetOnChangedFinal().Insert(OnDeadZone);
    }

    protected void OnDeadZone(SCR_SliderComponent s, float v)
    {
        MyDriveSettings.Set("m_fHeadingDeadZone", v);
    }
}

```

Crew turrets on a moving torso

Gunner, commander and driver turrets are slots on the **root** (they must be, for compartments and replication). The torso they sit in is a kinematic part that turns and crouches. Two tools keep them together:

- `EntitySlotInfo.SetAdditiveTransformLS()` moves the whole turret with the torso (translation only).
- A **seat bone** in the turret model gets the torso rotation. Moving a *bone* keeps the character seated; moving the passenger point every frame made characters stand up.

EXAMPLE CODE - FREE TO USE

```

void FollowTorso(EntitySlotInfo slot, IEntity turretEnt, vector torsoDelta[4])
{
    // 1) translate the turret with the torso
    vector add[4];
    Math3D.MatrixIdentity4(add);
    add[3] = torsoDelta[3];
    slot.SetAdditiveTransformLS(add);

    // 2) rotate the seat via a bone, not via the passenger point
    Animation anim = turretEnt.GetAnimation();
    TNodeId bone = anim.GetBoneIndex("v_seat");
    if (bone == -1) // TNodeId is an id, not an index: negative values are valid, only -1 is
        "none"
        return;

    vector m[4];
    Math3D.MatrixCopy(torsoDelta, m);
    m[3] = m_vSeatRest;
    anim.SetBoneMatrix(turretEnt, bone, m);
}

```

PITFALL

Pause bone moves while a character gets in or out

(`CompartmentAccessComponent.IsGettingIn()` / `IsGettingOut()`) and for a moment after – otherwise the get-in animation aims at a moving target.

Who drives what

SEAT	CONTROLS	NOTES
Driver turret	walking heading, park, jump	stabilized in the world, never drives the torso
Gunner turret	torso yaw, arm pitch, arm weapons	the seat turns to face the aim direction
Commander turret	shoulder launchers, targeting, modules	launcher yaw relative to the torso, limited

Seat rules keep AI and automation in line: the driver turret is never offered to AI, the vanilla pilot seat is only for AI drivers, and it is blocked while a player drives from the turret.

The mod at a glance

What all the techniques on this site add up to: the M.A.W.S. walker, its weapons, modules and controls. Click an image to see it full size.

System overview

M.A.W.S. SYSTEM OVERVIEW

ARMA REFORGER - VEHICLE MOD

M.A.W.S.

MECHANIZED ASSAULT WALKER SYSTEM

A 14-metre, three-crew assault walker with five modular hardpoints. Fit it for anti-armour, fire support or air defence, then take it where tracks cannot go.

14 m

HEIGHT

3

CREW

5

HARDPOINTS

20 t

MAX. PAYLOAD

60 km/h

TOP SPEED, EMPTY

2 × 12.7 mm

FIXED MG PER ARM

MECHANIZED ASSAULT WALKER SYSTEM

STATION 1

DRIVER

Walks with W/S, side steps with A/D. The view sets the walking direction. Hop, jump jets, park.

STATION 2

GUNNER

Aims the torso and fires both arms. Stabilised optics from 1× to 25×, thermal with the sensor mast.

STATION 3

COMMANDER

Fire control: marks targets, builds the fire plan and launches the shoulder weapons. Feeds the datalink.

ON THE MOVE

- Top speed 60 km/h empty, -1 km/h per tonne of payload
- Hop about 3.7 m, double tap for jump jets
- Infantry in sight shows as dots: hostile, unknown, friendly



XM-29 WARDEN

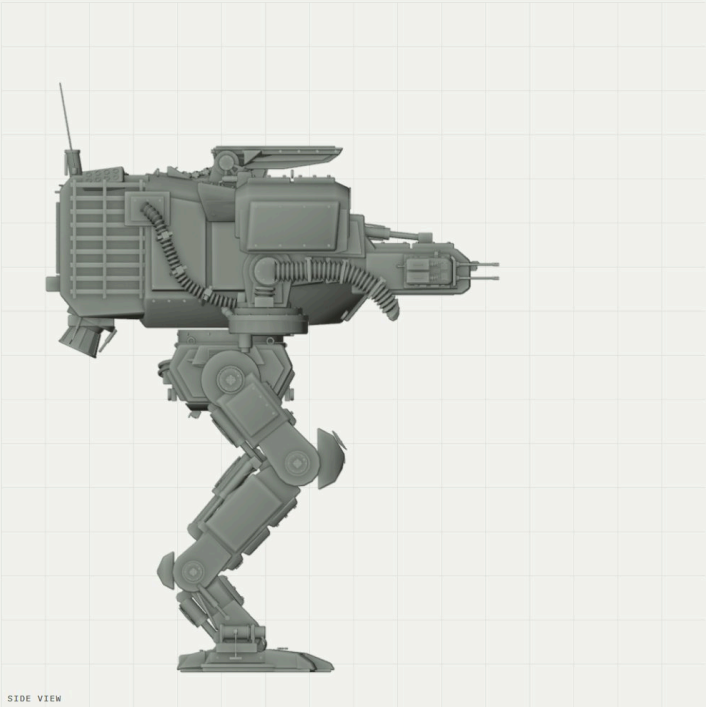
United States - digital woodland



SHBM-29 ZUBR

Soviet Union - fine digital flora

SIDE VIEW



Three crew stations, five hardpoints, 60 km/h empty. The [crew seats](#), [leg drive](#) and [loadout](#) pages explain how each part works.

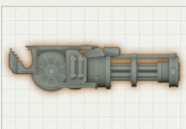
Arm weapons

M.A.W.S. · ARM HARDPOINTS

MECHANIZED ASSAULT WALKER SYSTEM

ARM WEAPONS

Each arm carries a fixed twin 12.7 mm machine gun plus one module. LMB RT fires, R X switches ammunition.



ROTARY CANNON

GATLING 30 MM

USSR **GSh-6-30** · 4.5 t · ~4.5 km/h

Six-barrel rotary cannon for infantry, light vehicles and helicopters. Spins up in 0.4 s and holds a wall of fire while the gunner tracks.

RATE OF FIRE	1,800 rds/min
AMMUNITION	1,200 HE + 1,200 AP
SPIN-UP	0.4 s
DISPERSION	2.0 mrad

LMB Fire arm weapons
R Switch HE / AP



MAIN GUN

CANNON 120 MM

USSR **2A82 125 mm** · 6 t · ~6 km/h

Smoothbore gun with autoloader on the arm. The barrel recoils into the mount on every shot. Carries kinetic and shaped-charge rounds.

AMMUNITION	12 HEAT-MP + 12 APFSDS
RELOAD	4 s autoloader
RATE OF FIRE	15 rds/min
DISPERSION	0.3 mrad

LMB Fire arm weapons
R Switch HEAT / APFSDS



KINETIC PENETRATOR

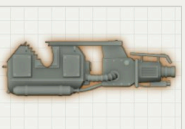
RAILGUN

USSR **Railgun** · 5.5 t · ~5.5 km/h

Electromagnetic launcher firing a 2.5 kg slug at 2,500 m/s. Each shot drains the capacitor bank, and the rails are swapped at a depot.

MUZZLE VELOCITY	2,500 m/s
PROJECTILE	2.5 kg slug
CAPACITY	4 shots
CAPACITOR CHARGE	8 s

LMB Fire arm weapons



AREA DENIAL

FLAMETHROWER

USSR **Flamethrower** · 3 t · ~3 km/h

Pressurised fuel projector for close assault on trenches, treelines and buildings. Burning ground keeps burning after the stream stops.

REACH	70 m cone
FUEL	about 5 min continuous
EFFECT	Afterburn on ground

LMB Fire arm weapons

FIXED TWIN 12.7 MM

Every arm carries a twin heavy machine gun in addition to its module: 550 rounds per minute, 1,000 rounds of each of two ammunition types.

WEAPON GROUPS

Hold V / D-pad – to switch between machine guns and modules, tap to change the fire mode. The HUD weapon block shows ammunition, reload and charge state per arm.

PAYLOAD

Arm modules weigh 3 – 6 t. Every tonne costs 1 km/h of top speed; above 20 t of total payload the walker is overloaded and throttled further. USSR walkers carry Soviet equivalents.

Shoulder weapons

M.A.W.S. · SHOULDER HARDPOINTS · FIRE CONTROL

MECHANIZED ASSAULT WALKER SYSTEM

SHOULDER WEAPONS

Mark targets with I, build a fire plan, launch with N. Reloads need a standstill near support, or the ammo pack.



70 MM ROCKETS

ROCKET POD

USSR **S-8 80 mm** · 3 t · ~3 km/h

21-cell pod with unguided HE, guided and cluster rockets. Guided rockets fire at any range and are the only weapon that can track aircraft.

CELLS	21 per pod
AIR TARGETS	up to 4 km, guided only
CLUSTER	14 submunitions, min. 1,000 m
RELOAD	30 s standing still

I Mark target
N Fire salvo
- / + Salvo size + / -
I Ammunition type



GUIDED ANTI-TANK MISSILE

HELLFIRE

USSR **Kornet** · 1.8 t · ~1.8 km/h

Four laser-guided missiles on rails. They follow the walker's own laser spot or a friendly designator. Hellfire does not engage aircraft.

RAILS	4
RANGE	100 – 8,000 m
PROFILE	top attack from 400 m
RELOAD	60 s standing still

I Mark target
N Fire salvo
Insert Next target



INDIRECT FIRE

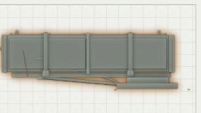
MORTAR

USSR **2B14 82 mm** · 1.5 t · ~1.5 km/h

Shoulder mortar for indirect fire beyond line of sight. Charge is computed for the range. Targets come from the map or from a mark.

RANGE	150 – 6,000 m
MAGAZINE	24 HE + 12 smoke
INTERVAL	2 s
RELOAD	30 s standing still

I Mark target
N Fire salvo
I HE / smoke



HEAVY STRIKE MISSILE

APOCALYPSE

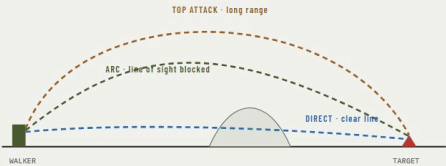
USSR **APOCALYPSE** · 8 t · ~8 km/h

Heavy cruise-missile box with four cells. It launches at 35° and dives on the target from 250 – 700 m altitude. The heaviest single module.

CELLS	4 per box (2 × 2)
RANGE	300 – 4,000 m
PROFILE	top attack from 600 m
RELOAD	90 s standing still

I Mark target
N Fire salvo

FLIGHT PROFILES



FIRE CONTROL

ROCKET POD, GUIDED	no minimum range - aircraft up to 4 km
CLUSTER ROCKETS	minimum 1,000 m
HELLFIRE	100 – 8,000 m - ground targets only
MORTAR	150 – 6,000 m - ballistic
APOCALYPSE	300 – 4,000 m

Before launch the walker checks the line of sight and picks a profile per missile: top attack at long range, a direct line when the target is visible, an arc over cover when it is not. Only guided 70 mm rockets track aircraft. With the sensor mast, Hellfire follows the walker's own laser spot.

The fire control and lock logic behind these is described in [Targeting system](#).

Back modules

M.A.W.S. BACK HARDPOINT · SURVIVABILITY

MECHANIZED ASSAULT WALKER SYSTEM

BACK MODULES

One back slot for sensors, network, supply or protection. Every walker carries smoke and decoys as standard.



SENSOR MAST

LASER / THERMAL MAST

USSR **Laser / thermal** · 1 t · -1 km/h

Raised sensor head with laser designator and thermal sight. Triples the aiming range and guides the walker's own Hellfire missiles.

AIR RANGE	6,000 m (2,000 m without)
LASER	own spot, 4 updates/s
THERMAL	3

- J** Thermal on / off
I Mark target



NETWORK ANTENNA

DATALINK

USSR **Datalink** · 0.8 t · -0.8 km/h

Shares what the walker sees. Vehicle tracks and infantry groups in line of sight appear on the map of every friendly player.

VEHICLES	map markers, refreshed every 4 s
INFANTRY	1,500 m, groups within 60 m
WITHOUT IT	NO DATALINK – VISUAL ONLY

Passive. Works while fitted and undamaged.



FIELD RELOAD

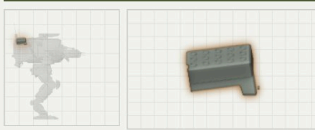
AMMO PACK

USSR **Ammo pack** · 2 t · -2 km/h

Back-mounted magazine that refills the launchers away from support. Each charge replaces one depot visit. Refilled at support vehicles.

CHARGES	2 full reloads
USE	stand still, launchers refill
REFILL	at supply / depot

Passive. Works while fitted and undamaged.



SMOKE AND DECOYS

COUNTERMEASURES

USSR **Countermeasures** · 0.5 t · -0.5 km/h

Every walker carries built-in smoke and decoy launchers; this module carries four times the standard load. Each burst lays a smoke screen towards the threat and decoys guided missiles. AUTO is the default.

STANDARD	16 smoke + 48 flares + 48 chaff, always fitted
WITH MODULE	x 4: 64 smoke + 192 flares + 192 chaff
BURST	4 smoke + 4 flares + 2 chaff
SMOKE SCREEN	25 m out, 70° arc, 60 s
WARNING	missiles and RPGs, 3,000 m
AUTO	fires 4.5 s before impact

- Delete** Fire countermeasures
Page Down AUTO / MANUAL

DAMAGE MODEL

- Small arms have no effect on the armour; anti-tank weapons and cannons do.
- 14 hit zones. A destroyed leg drops the walker into a crouch with a list.
- Damaged hydraulics slow every movement; a destroyed zone disables its modules.

HIT POINTS

Hull	12,000	Arms	3,000
Cockpit	3,000	Shoulders	2,500
Reactor	4,000	Back	2,500
Hydraulics	2,500	Sensors	1,500
Legs	3,000	Jets	2,000

REPAIR

50 %

FIELD REPAIR

100 %

DEPOT / GAME MASTER

Controls

M.A.W.S. DEFAULT CONTROLS

MECHANIZED ASSAULT WALKER SYSTEM

CONTROLS

Standard Reforger vehicle keys where they exist. Everything can be rebound under Settings > Controls.

MOVEMENT · DRIVER

W / S	Walk forward / backward
A / D	Side step
Mouse	Walking direction (driver turret)
Space	Hop
Space x 2	Jump jets
End	Park / crouch

ARM WEAPONS · GUNNER

LMB	Fire arm weapons
R	Switch arm ammunition
RMB	Zoom
Alt + wheel / Num + -	Zoom level 1x – 25x
Alt	Free look

MISSILES · COMMANDER

I	Mark target
Insert	Next target
Home	Clear fire plan
N	Fire salvo
- / .	Salvo size + / -
/ (DE: -)	Ammunition type

SYSTEMS

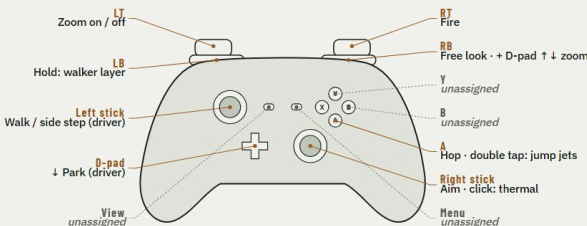
U	Module menu
J	Thermal
Delete	Smoke + flares + chaff
Page Down	Countermeasures AUTO / MANUAL

HOLD LB / L1 · WALKER LAYER

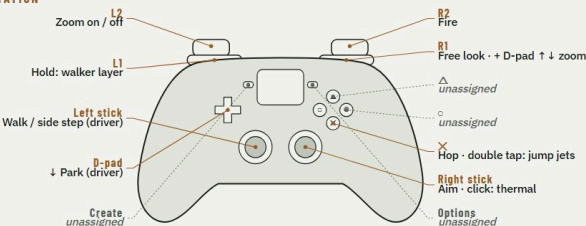
XBXX	PS	ACTION
RT	R2	Fire salvo
A	X	Mark target
X	△	Next target
Y	□	Ammunition type
D-pad ↑ / ↓	D-pad ↑ / ↓	Salvo size + / -
D-pad ←	D-pad →	Clear fire plan
LS click	L3	Module menu
RS click	R3	Smoke + flares
Hold RS	Hold R3	Countermeasures AUTO / MANUAL

While LB is held, RT, X and D-pad – do not fire or switch the arm weapons.

XBXX



PLAYSTATION



All keys can be rebound under *Settings* → *Controls* → *GTG Walker*; driver steering dead zone and sensitivity are adjustable there too.

M.A.W.S. License

Version 1.0 – © 2026 Seebie90. What you may do with the M.A.W.S. mods, with the example code and with this documentation.

This license covers the mods **M.A.W.S. – Mechanized Assault Walker System** and **M.A.W.S. Sci-Fi Expansion Pack** for Arma Reforger ("the Mod"), including all scripts, prefabs, 3D models, textures, materials, sounds, configs, UI layouts and other files they contain. All rights reserved, except as granted below.

1. You may, without asking

- **Use the Mod** – play it, run it on your servers, use it in your own scenarios and missions.
- **Make retextures** (new textures or material variants) and publish them as a separate mod, if
 - the original Mod stays a **dependency** – your mod contains no files of the Mod itself,
 - your description **credits the author** ("M.A.W.S. by Seebie90") and links to the original Mod,
 - the **3D models are not changed**.

2. You may, only with prior approval

Extensions – mods that add to or change the behaviour of the Mod (new modules, weapons, loadouts, scripts that hook into or mod its classes). Ask the author **before** you publish; approval must be given in writing (for example via Discord) and is at the author's discretion. Approved extensions keep the original Mod as a dependency and credit the author as in section 1.

3. You may not

1. **Reupload** the Mod or any part of it – scripts, prefabs, 3D models, textures, sounds, configs or layouts, whole or in parts, in any form (including repacks, merged mods and "fixed" or "updated" copies).
2. **Modify the 3D models** of the Mod or distribute modified versions of them.
3. **Copy code from the Mod** into your own mods, including code extracted from the published files.
4. **Remove or hide the credits** of the author.
5. **Sell** the Mod or charge for access to it or any part of it.

4. Example code on this site

The code examples on this site were written for teaching and are separate from the Mod's code. You may use, change and publish them freely in your own projects. Credit is appreciated, not required. This covers only the examples shown on this site, not the Mod itself.

5. This documentation

- You may read it online and download the PDF for your **personal** use.
- You may **not** reproduce it in printed form, modify its content, or republish it (in whole or in parts) on other sites or platforms. Linking to it is welcome.
- **Suggestions and corrections** go to the author via the Discord forums.

6. Other terms

- The Bohemia Interactive Arma Reforger EULA and Workshop terms always apply in addition to this license.
- Breaking this license ends any permission granted by it; the author may request the removal of mods that break it.
- For anything not covered here, contact the author.

Contact: Discord **Seebie90 I Nomad** – discord.gg/gtg-german-tactical-group

Deutsche Kurzfassung

Unverbindlich – maßgeblich ist der englische Text.

- **Nutzen** (spielen, Server, eigene Szenarien): frei.
- **Retexturen** erlaubt, wenn die Original-Mod Abhängigkeit bleibt, der Autor genannt und verlinkt wird und die 3D-Modelle unverändert bleiben.
- **Erweiterungen** nur mit vorheriger Genehmigung.
- **Verboten:** Reuploads (Skripte, Prefabs, Modelle, alle Dateien, auch in Teilen), geänderte 3D-Modelle, Code übernehmen, Credits entfernen, Verkauf.
- **Code-Beispiele** auf dieser Seite sind frei nutzbar.
- **Diese Doku:** online lesen und das PDF für den eigenen Gebrauch laden ja – keine gedruckte Vervielfältigung, keine Veränderung, keine Neuveröffentlichung. Vorschläge über die Discord-Foren an den Autor.

Extending a mod without touching it

The Sci-Fi Expansion Pack adds an energy shield, a gravity field and more to M.A.W.S. – as a separate add-on that only uses `modded class` and a few hooks the base mod exposes.

IN PLAIN WORDS

The add-on changes the base mod's classes from outside (modded class). The base mod keeps a few empty methods ready that the add-on fills in.



SCI-FI EXPANSION

Three new modules and three systems built into every walker. **The AI crew uses all of it.** Requires M.A.W.S.



ARM MODULE · ENERGY WEAPON

LASER DISRUPTOR

Arm · 5 t · ~5 km/h

Continuous deep-red beam, no ammunition: capacitors charge back on their own. Short bursts hit hardest, long fire weakens down to a floor. Sets vehicles on fire.

RANGE	800 m
DAMAGE	continuous, scales with charge (min. 30 %)
CAPACITORS	8 s beam · recharge 12 s
VS. SHIELDS	stopped at the bubble – drains it hardest

LMB Hold to fire the beam

BUILT INTO EVERY WALKER

ENERGY SHIELD

- 18,000 points, recharges 5 s after the last hit, full in 20 s
- Something always gets through: holds **95 %** of energy weapons (they cost it double), **85 %** of blasts, **75 %** of kinetic rounds
- Also protects the crew · collapses for 10 s at 0 % · HUD bar under the compass



ARM MODULE · AREA WEAPON

AGL40 IMPLOSION

Arm · 4 t · ~4 km/h

40 mm grenades that open a singularity: everything within 30 m is pulled into the centre for 2 s, then blown outwards.

MAGAZINE	16 grenades · 1 every 3 s
VELOCITY	240 m/s, high arc
PULL RADIUS	30 m · 2 s, then blast
IMMUNE	vehicles over 15 t, walkers

LMB Fire grenade

ROCKET POD AMMUNITION

EMP ROCKETS

- Guided 70 mm rocket – cycle to EMP with the ammo key
- 30 m burst: night vision and holo sights dead for **60 s**, vehicle engines for **2 min**
- Against walkers: ~40 % shield per burst – 2 to 3 rockets break it



BACK MODULE · FIELD GENERATOR

GRAVITY DISTORTION

Back · 3 t · ~3 km/h

Charges for 5 s, then lifts everything around the walker 10–30 m into the air for 15 s – and slams it down at 4 g. Hits friend and foe; the field moves with the walker.

RADIUS	60 m
CYCLE	5 s charge · 15 s lift · 2 s slam
COOLDOWN	90 s
SHIELD	off during the field, then recharges

Ctl1+6 Activate (any crew seat)

COMMANDER · MAP

VEHICLE HACKING

- Click a vehicle on the map → **HACK** (60 s – 3 min by weight and firepower, max. 2 km)
- Then **HACK MENU**: Engine off · Unload all weapons · Full power · Overtake (AI crews switch sides for 5 min)
- One try per option, chance by difficulty · hacked vehicles get a blue hexagon on the map

Three new modules and three systems built into every walker. The AI crew uses all of it.

Systems

How an add-on hooks in

The base mod exposes a small extension singleton with empty virtual methods. The add-on mods that class and overrides only what it needs. Without the add-on, the base mod runs its defaults – no checks, no `#ifdef`, no load-order tricks.

EXAMPLE CODE – FREE TO USE

```
// --- base mod
class MyWalker_Extension
{
    protected static ref MyWalker_Extension s_Instance;

    static MyWalker_Extension Get()
    {
        if (!s_Instance)
            s_Instance = new MyWalker_Extension(); // with an add-on loaded this creates the
MODDED class
        return s_Instance;
    }

    //! Damage that reaches the zones after armour. Return what is left.
    float FilterWalkerDamage(MyWalker_DamageModel model, float damage, BaseDamageContext ctx)
    {
    }
```

```
        return damage;
    }
}

// --- add-on
modded class MyWalker_Extension
{
    override float FilterWalkerDamage(MyWalker_DamageModel model, float damage,
BaseDamageContext ctx)
    {
        damage = super.FilterWalkerDamage(model, damage, ctx);
        return model.MyShield_Absorb(damage, ctx);
    }
}
```

NOTE

`modded class` replaces the class everywhere, including `new` calls inside the base mod. That is why a plain singleton is enough as an extension point.

State lives where it already replicates

New replicated state is added to components that already exist on the walker (damage model, module manager) via `modded class` with extra `[Rp1Prop]` fields. No new components in the base prefab, no prefab edits in the add-on.

Energy shield

18,000 points around every walker. Something always gets through, energy weapons hit it hardest, and it collapses for 10 seconds when it runs dry.

IN PLAIN WORDS

Before damage reaches the walker, the shield takes most of it out of the total and loses points for it. Mines placed under the walker are not stopped.

Filter, not a hit zone

The shield is not a collider. It sits inside the walker's damage model, after armour, and counts in the same points as the zones. That makes balancing simple: an RPG hit is ~1,300 points whether it lands on the shield or the leg.

DAMAGE TYPE	SHIELD HOLDS	SHIELD COST
Energy (laser)	95 %	x2 per point held
Explosive / fragmentation / fire	85 %	x1
Kinetic (APFSDS, autocannon, minigun)	75 %	x1
EMP	-	40 % of maximum, direct

EXAMPLE CODE - FREE TO USE

```
///! [Server] Returns the damage that still reaches the walker.
float MyShield_Absorb(float damage, BaseDamageContext ctx)
{
    if (damage <= 0 || m_fShieldPoints <= 0)
        return damage;

    m_fLastHit = Now();                // regeneration waits 5 s after the last hit

    float hold = HOLD_KINETIC;
    float cost = 1.0;
    if (s_bEnergyHit)                  // set by the laser around its damage call
    {
        hold = HOLD_ENERGY;
        cost = COST_ENERGY;
    }
    else if (IsExplosive(ctx.damageType))
        hold = HOLD_EXPLOSIVE;
```

```

    if (IsPlacedChargeInside(ctx))           // mines / C4 under the walker: no protection
        return damage;

    float absorbed = Math.Min(damage * hold, m_fShieldPoints / cost);
    m_fShieldPoints -= absorbed * cost;
    if (m_fShieldPoints <= 0.5)
    {
        m_fShieldPoints = 0;
        m_fDownUntil = Now() + 10;          // collapsed
    }

    SyncShield();                            // replicated percent + hit counter (for the flash)
    return damage - absorbed;
}

```

Inside the bubble

A rocket explodes *at* the walker – inside the bubble – but it flew in from outside. A mine was already inside. The difference is the damage **source**: anything without a projectile move component that sits inside the bubble ellipsoid gets no protection.

EXAMPLE CODE - FREE TO USE

```

bool IsPlacedChargeInside(BaseDamageContext ctx)
{
    IEntity src = ctx.damageSource;
    if (!src || src.FindComponent(ProjectileMoveComponent))
        return false;           // projectile / rocket / grenade: came from outside

    vector o = GetOwner().CoordToLocal(src.GetOrigin()) - BUBBLE_CENTRE;
    vector r = BUBBLE_RADII;     // ellipsoid half axes
    float e = (o[0] / r[0]) * (o[0] / r[0]) + (o[1] / r[1]) * (o[1] / r[1]) + (o[2] / r[2]) *
(o[2] / r[2]);
    return e < 1.0;
}

```

Beams stop at the bubble

Hitscan beams (the laser) do not create a damage context with a projectile – they trace and apply damage themselves. To make them stop at an enemy shield, the beam is intersected with every standing bubble first: ray against ellipsoid, solved in the walker's local frame scaled to a unit sphere.

EXAMPLE CODE - FREE TO USE

```

//! Distance along the ray to the bubble, or -1.
float RayVsBubble(IEntity walker, vector start, vector dir, float range)
{
    vector o = walker.CoordToLocal(start) - BUBBLE_CENTRE;
    vector d = walker.VectorToLocal(dir);
    vector r = BUBBLE_RADII;
}

```

```

vector os = Vector(o[0] / r[0], o[1] / r[1], o[2] / r[2]);
vector ds = Vector(d[0] / r[0], d[1] / r[1], d[2] / r[2]);

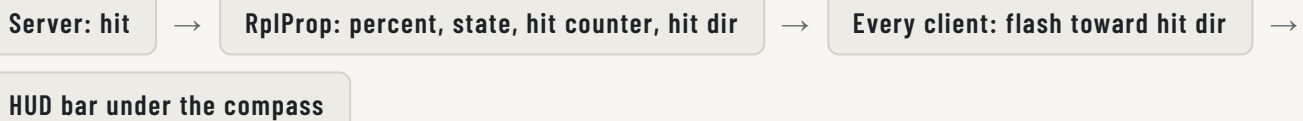
float a = vector.Dot(ds, ds);
float b = 2.0 * vector.Dot(os, ds);
float c = vector.Dot(os, os) - 1.0;
if (c < 0)
    return -1;           // starts inside (own walker): no interception

float disc = b * b - 4.0 * a * c;
if (disc < 0)
    return -1;
float t = (-b - Math.Sqrt(disc)) / (2.0 * a);
if (t < 0 || t > range)
    return -1;
return t;
}

```

The bubble effect

The visible bubble (hexagon grid + haze) is a local child entity spawned on every machine with a display – never replicated. It shows while the shield stands and flashes in the direction of each hit, driven by a replicated hit counter and direction. The dedicated server spawns nothing.



NOTE

Crew protection: some weapons damage the crew characters directly (e.g. a railgun passing through). A modded character damage manager asks the walker's shield first and charges it for what it absorbs.

Gravity field

A back module that lifts everything around the walker 10–30 m into the air, holds it there for 15 seconds and then slams it down at four times gravity.

IN PLAIN WORDS

The server switches the field through its phases. Every machine lifts or slams the objects whose physics it simulates itself.

Phases

PHASE	DURATION	WHAT HAPPENS
CHARGE	5 s	boundary effect, warning sound
LIFT	15 s	every object with mass in the circle floats up to its hover height and drifts slowly
SLAM	2 s	extra 3 g downwards (= 4 g total); characters are guided down and hit the ground
IDLE	90 s cooldown	shield recharges, which was off during the field

Only the server switches phases. Phase, a phase counter and the remaining cooldown are replicated through the module manager (via `modded class`).

Every machine moves what it simulates

Physics of a player-driven car runs on that player's client, AI characters run on the server, dropped items may run anywhere. Applying forces only on the server would miss most of them. Instead every machine applies the field to the bodies **it** simulates.

EXAMPLE CODE - FREE TO USE

```
///! Per frame on every machine while LIFT or SLAM (callqueue: vehicle frames do not tick on foreign clients).
protected void FieldTick()
{
    float dt = GetGame().GetWorld().GetTimeSlice();
```

```

foreach (IEntity e : m_aInField)
{
    if (!e)
        continue;

    RplComponent rpl = RplComponent.Cast(e.FindComponent(RplComponent));
    if (rpl && !rpl.IsOwner())
        continue;           // someone else simulates it

    Physics p = e.GetPhysics();
    if (!p || !p.IsDynamic())
        continue;

    if (m_iPhase == PHASE_LIFT)
        Hover(e, p, dt);
    else
        p.ApplyImpulse(Vector(0, -3.0 * 9.81 * p.GetMass() * dt, 0));    // +3 g
}
}

```

Hover as a spring

A plain upward force would launch light objects into orbit and barely move heavy ones. A damped spring toward a per-object hover height (start height + 8..12 m, with a slow pendulum) works for crates and cars alike.

EXAMPLE CODE - FREE TO USE

```

protected void Hover(IEntity e, Physics p, float dt)
{
    MyGravObj o = m_mObjects.Get(e);
    o.m_fPhase += dt * Math.PI2 / o.m_fPeriod;
    float target = o.m_fHover + 0.6 * Math.Sin(o.m_fPhase);

    vector v = p.GetVelocity();
    float err = target - e.GetOrigin()[1];
    float vy = Math.Clamp(err * 1.5, -6, 6);           // m/s toward the hover height
    v[1] = v[1] + (vy - v[1]) * Math.Clamp(dt * 3, 0, 1);
    v[0] = v[0] + (o.m_vDrift[0] - v[0]) * Math.Clamp(dt, 0, 1);
    v[2] = v[2] + (o.m_vDrift[2] - v[2]) * Math.Clamp(dt, 0, 1);
    p.SetVelocity(v);
    p.SetAngularVelocity(o.m_vSpin);
}

```

Characters

Characters do not respond to forces while animated, and their ragdoll bones cannot be reached from script. What works: the server makes every living character in the field **unconscious** – a vanilla state that puts it into ragdoll and replicates cleanly. The flight path is simulated in script and written as a **position**

with `Teleport` ; the ragdoll follows and tumbles on its own. Survivors wake up three seconds after the field.

EXAMPLE CODE - FREE TO USE

```
// server, once per character that enters the field
SCR_CharacterControllerComponent cc =
SCR_CharacterControllerComponent.Cast(ch.GetCharacterController());
if (cc && cc.GetLifeState() == ECharacterLifeState.ALIVE)
    cc.SetUnconscious(true);           // ragdoll; woken again after the field

// every frame: AI characters on EVERY machine (same random seed), player characters only on
// their own client
vector m[4];
ch.GetWorldTransform(m);              // keep the body's own orientation
m[3] = o.m_vSimPos;                   // set only the position
ch.Teleport(m);
```

PITFALL

`PhysicsRagdoll.GetRagdoll()` returns null for characters – the bones cannot be driven. Work with the character root only.

PITFALL

Do not rebuild the transform from the current one plus a little extra rotation every frame: the ragdoll already turns the root, the two add up and the body spins faster and faster. A fixed upright orientation, on the other hand, pins the ragdoll upright. Write the position only.

PITFALL

If only the server teleports AI characters, their ragdolls fall a little between two network packets on every client and jerk up and down. Every machine computes the same path from the same seed and sets it itself.

The slam: guided, not dropped

Both obvious solutions failed in the test. Teleporting ragdolls downward at 35 m/s flung one 24 km away and crashed the client ("Entity out of world bounds"). A free fall from 30 m reaches over 20 m/s, and thin ragdoll limbs tunnel through the terrain. The working version keeps the 4 g but caps the speed and slows down near the ground. The impact damage is still that of a 4 g fall from the hover height, credited to whoever triggered the field.

EXAMPLE CODE - FREE TO USE

```
sv[1] = sv[1] - 4 * 9.81 * dt;
float maxFall = Math.Clamp((sp[1] - ground) * 2.5, 3, 12); // at most 12 m/s, slower near the
```

```

ground
if (sv[1] < -maxFall)
    sv[1] = -maxFall;

sp = sp + sv * dt;
if (sp[1] <= ground + 0.2)
    Land(ch); // stop teleporting; damage after 0.3 s

// damage: what a 4 g impact from the hover height would do
float impact = Math.Sqrt(2 * 4 * 9.81 * hoverHeight);
float damage = Math.Max(0, impact - 6) * 12;
MyDamage.Apply(ch, EDamageType.KINETIC, damage, Instigator.CreateInstigator(triggeredBy));

```

PITFALL

Anything pushed below the terrain by physics (a dropped rifle under 4 g) can crash the server with an "out of world" assert. A small fall guard watches the area for a few seconds after the slam: below the terrain the server deletes the object, a client lifts a replicated one back up and waits for the server's correction.

NOTE

Other walkers are excluded on purpose: their legs and 50 t of mass are driven by script, a second force source would make them fight each other.