

LOW POLY

# war pack

by polyperfect



*Have a Suggestion?*

[info@polyperfect.com](mailto:info@polyperfect.com)

# Thanks!

First of all, thank you for purchasing our pack, we really appreciate that! We are putting a lot of effort into this.

We are also planning to expand the list of the characters and their animations in the future with free updates of the pack. Check out our [Facebook Page](#) for any news.

# Handy Links ;)

## Other Low Poly Packs

[Low Poly Animated Animals](#)

[Low Poly Animated People](#)

[Low Poly Animated Dinosaurs](#)

[Low Poly Animated Prehistoric Animals](#)

[Low Poly Epic City](#)

[Low Poly Ultimate Pack](#)

[Low Poly War Pack](#)

## Toolkits

[Ultimate Crafting System](#)

## 2D Packs

[Low Poly Icon Pack](#)

[Fancy Icon Pack](#)

## Follow us

[Facebook](#)

[Instagram](#)

[Youtube](#)

[Polyperfect.com](#)

# Updates

## **VERSION 3.15**

Updated common scripts and relocated some files to improve compatibility with Crafting System

## **VERSION 3.10**

We have to rewrite some of the old code to make sure the pack will now work with any of our new packs flawlessly. Unfortunately with this one, there isn't a simple update procedure--the old war pack in the project needs to be deleted before importing the new version. Trying to update an existing version seems to cause random unity errors, even crashes depending on the version. So if you don't need to have the crafting pack in the same folder etc., please don't update.

## **VERSION 3.7**

Some minor fixes

## **VERSION 3.0**

WE WANTED TO PREPARE A REGULAR UPDATE AND INSTEAD OF IT WE ACCIDENTALLY CREATED A WHOLE GAME FRAMEWORK 🌐

**Unified Enemy and Ally behavior:** Any feature added to one can be added to the other without additional code. Units now belong to factions, which can be allied or hostile to each other.

**Updated Vehicles:** Vehicles will now be targeted and shot by opposing factions and can be destroyed without also destroying the player and camera.

**Modularized scripts:** Variants can be created by adding, removing, disabling, or otherwise altering the components on a GameObject. Removing a GameObject or script does not cause all behaviors to break anymore. All units with Health use the same Health script, etc.

**In-editor documentation:** Every script has help boxes describing their use and noting which components are required. Mistakes in setup are highlighted in red.

**Extensible scripts:** Coding new behaviors has a clear path to avoid conflicts with other code files.

**Observant UI:** UI scripts observe and register with their targets instead of being driven by them, so units don't need UI to function and won't be broken if the UI is removed.

**Fuzzy Logic System:** AI Soldiers now care about ammo and health, and weapon pickups. They will seek out resources they are low on. They hold grudges even against allies and will seek out enemies or an objective.

**Many event callbacks:** Sound effects, animations, visual effects, and other things can be easily added or changed without altering any code. When writing custom code, these callbacks can be used so new behaviors use, instead of depending on, each other.

**Nested Prefab and Variant based system:** New behaviors and sound effects can be added to a base, and all prefabs that contain it or are variants of them will be updated, instead of having to copy and paste it to all and remember which things to put where.

**Domain Reload-less Play Mode:** The new scripts all support Enter Play Mode Options, reducing Enter Play Mode time by 95% or more for faster iteration.

### **VERSION 2.0**

New scripts, new models, new characters. New everything!

We have remade the pack from the ground up. All the scripts, animations, rigs, and models were changed so please, DO NOT UPDATE from the older versions. Also it is worth mentioning that there will probably be another smaller update to balance and polish things out. We hope you will like our work, we have put LOT of effort and love into this small pack.

### **VERSION 1.25**

Bug fixes and tweaks

# Modular Scripts

## **What are modular scripts?**

Modular scripts break large problems down into smaller parts. Instead of having a single Player script that controls everything a player does, the player's avatar is just a normal soldier that also has a PlayerInputs script attached. Likewise, the NPC soldiers simply have AI scripts attached, which send inputs to the soldier in the same way that the PlayerInputs script does, just by using artificial intelligence instead.

## Why use modular scripts?

A traditional problem with the old approach (just having a single Player script for example) is that changing behavior at all requires editing code. And in doing so, anything that used the old code could break. This can make it slow and frustrating to add new things to your game. Modular scripts on the other hand work with each other, rather than depending on each other. This means you can add or remove one bit of behavior and the rest will still work fine, without having to edit any code. And if you do have to add code, you don't need to change (and thus risk breaking) existing code.

# Prefab Variants

## What are Variants?

Another strength of modular scripts is their inherent support for prefab variant-driven systems. Prefabs are simply preset GameObjects that you can create, and Variants are essentially “*This* Prefab, but with *these* changes”. For example, you might have a soldier that does everything an existing one does, but has a red material assigned instead of a green one. Doing this as a variant means that if you later change the way the original soldier works, the one with a red material will automatically update to match the first one. This can save significant time and headaches, especially over the course of larger projects.

## How are they used in the pack?

Prefab Variants are leveraged extensively in the project, with Player and AI-controlled soldiers being variants of a base soldier Prefab. Similarly, vehicles are

all set up in a Prefab Variant hierarchy, and their names reflect this; for example, the Sherman tank is called `Rideable_Tank_Sherman`, and is a variant of `Rideable_Tank`, which is a variant of the `Rideable` prefab. This makes it easy to keep track of which changes are coming from where.

# Events

The way that modular scripts handle interacting with, instead of depending on, each other is through the use of events. An event is something that happens at a particular time, and it can be used to trigger anything you want. For example, vehicles have an `OnEnter` and `OnExit` event that you can use to easily add sound effects or animations that you want. You can use these events, called `UnityEvents`, the same way you use Unity GUI Buttons.

# Usables

## **What is the Usables system?**

The Usables system is a part of the modular scripts system that makes it easy to use and respond to events. It handles specifying the use conditions and effects of...things.



A Usable can have Conditions, such as ammo or cooldown, and Effects, such as spawning a bullet or changing a camera.

## **How does one Use a Usable?**

A Usable can be used from code from any script, though the built-in way is through Usable Holders and Capabilities, the latter of which you can create in the Project view using Right Click > Create > Usable Capability. Soldiers, for example, have a Soldier\_UsableHolder script. This “holds” one of the Usables (Guns in the case of soldiers), and takes care of aiming and sending Shoot or Reload capabilities and the like. The Usable itself decides what to do with those capabilities.

## **Examples of Usables**

As mentioned previously, the Guns make heavy use of the Usables system, for both Shooting and Reloading. The vehicles also leverage them; when a player gets near a vehicle and presses E, it tries sending a RideEnter event. This RideEnter event is picked up by the vehicle, and contains information about the sender (the player). If the vehicle is happy with the player, it'll add the player to itself and let them know they've successfully entered. The player has a Rider script on it that then takes care of constraining the player to the vehicle, as well disabling renderers and colliders if applicable.

# Player

## Controls

The default player controls are noted below. The PlayerInputs script takes care of forwarding those inputs to the InputCollector component, which is read by the SoldierMovementFromInputs script to actually move.

### Keys

W - forward

S - back

A - left

D - right

E - interaction

SHIFT - sprint

RIGHT MOUSE - weapon aim

LEFT MOUSE - shoot

MOUSE WHEEL - switch weapon

# Soldiers

Soldiers are composed of a large number of scripts, each handling a specific task. The most important script for easy customization is the `SoldierMovementFromInputs` script. This has such parameters as the walk and run speed. There are many more scripts though--for example, they have a `Health_Reservoir` script which specifies their current and maximum health and provides events like `OnDeath` and `OnRevive` for other scripts to hook into. They also have various Carriers, such as an `Ammo_Carrier` and `Usable_Carrier`, which keep track of their inventory.

## AI

The soldiers make use of a Fuzzy Logic system, on the `FuzzyMachine` component. Fuzzy logic allows an entity to choose what action it should take depending on various parameters it's told to take into account. For example, the "Weight" of a medkit-seeking behavior increases as the health remaining decreases, making it more likely to be selected. Existing scripts ending in `"_AI"` or those you create inheriting from `FuzzyComponent` will all register with an attached `FuzzyMachine` automatically. There are multiple layers in a given `FuzzyMachine` which are handled by the script itself. The gunning layer for example chooses whether to shoot, reload, or do nothing, while the Movement layer tells the agent where it should move.

## Navigation

AI-driven soldiers will automatically work with Nav Meshes if they are present. To create a NavMesh, you need to bake the NavMesh of your scene. Open the Navigation window and in the bake tab set values and click Bake. These are our recommended values:

Inspector

Services

Asset Editor

Navigation

Agents

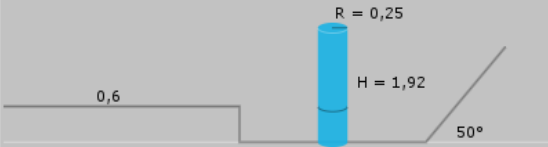
Areas

Bake

Object

[Learn instead about the component workflow.](#)

### Baked Agent Size



R = 0,25  
H = 1,92  
50°  
0,6

Agent Radius	0.25
Agent Height	1.92
Max Slope	<div><div></div>50</div>
Step Height	0.6

### Generated Off Mesh Links

Drop Height	0.3
Jump Distance	0.75

▼ Advanced

Manual Voxel Size	<input checked="" type="checkbox"/>
Voxel Size	0.08
	3,13 voxels per agent radius

Voxel size controls how accurately the navigation mesh is generated from the level geometry. A good voxel size is 2-4 voxels per agent radius. Making voxel size smaller will increase build time.

Min Region Area	2
-----------------	---

Height Mesh

☒

Clear

Bake

# Car Controller

The car controller is based on Unity wheel colliders.

## Set up

### Wheels

Here you can determine the number of axles in a size field. In each axle, you must select the right and left wheel collider.

### Motor

This is a Boolean, turn this on if you want this axle to be connected to the engine.

### Steering

This is a Boolean, turn this on if you want this axle to be able to turn.

### Brake

Turn this Boolean on if you want this axle to have brakes.

### Handbrake

Turn this Boolean on if you want this axle to be connected to the handbrake.

## Lights

Lights are divided into front and backlights. For each, you can add multiple unity lights.

### **Front Lights Renderer**

This is a link to the renderer of front lights mesh.

### **Front Lights On Material**

This is material that will be used on the Front Lights Renderer when you turn lights on.

### **Front Lights Off Material**

This is material that will be used on the Front Lights Renderer when you turn lights off.

### **Back Lights Renderer**

This is a link to the renderer of backlights mesh.

### **Back Lights On Material**

This is material that will be used on the Back Lights Renderer when you turn lights on.

### **Back Lights Off Material**

This is material that will be used on the Back Lights Renderer when you turn lights off.

## **Statistics**

### **Motor Torque**

This is torque in Newton-meters that will be applied on all axes with Motor boolean set to true.

### Brake Power

This is torque in Newton-meters that will be applied as brake power on all axes with Brake boolean set to true.

### Max Speed

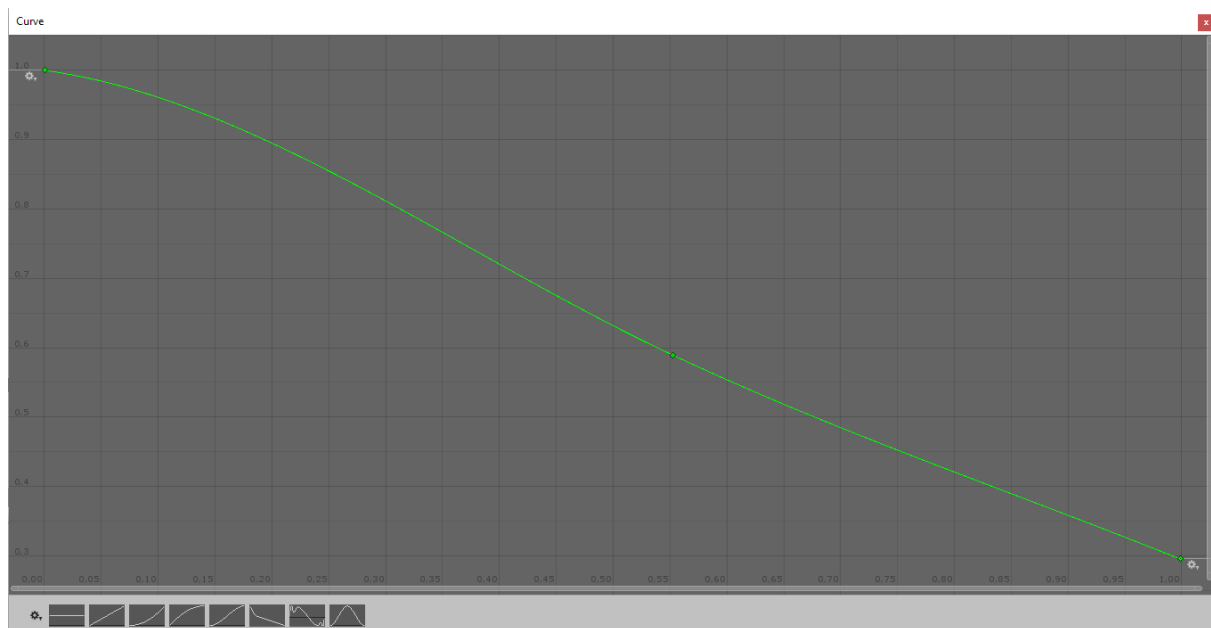
The maximum speed that a car can achieve.

### Max Steering Angle

Max steering angle of all wheels that are connected to the axle with Steering boolean set to true.

### Steering Curve

This curve determines how the Max Steering Angle is modified depending on the speed of the car.



On the horizontal axis is the speed of the car where 1 is the max speed of the car and 0 is 0 km/h.

On the vertical axis is the steering angle where 0 is 0 angle and 1 is the max steering angle.

### **Center Of Mass**

This sets the rigid body center of mass relative to the pivot point of the transform rigid body attached to.

### **Keys**

W - forward

S - back/brake

A - left

D - right

Space - handbrake

E - interaction

L - Lights On/Off

# **Tank Controller**

### **Set up**

The first tab is Set up. Here you can set up tank gun, belts, and other essential variables.

### **Belts**

Belts are controlled by bones that move dependent on wheel colliders.



### **Right Belt Renderer**

This is Skinned Mesh Renderer, which is necessary to simulate belt movement with bones.

### **Right Belt Wheels**

Here you set Wheel Colliders for the right belt, but only that is touching the ground. Plus for each, you select a bone that will have this wheel effect on.

### **Left Belt Renderer**

This is Skinned Mesh Renderer, which is necessary to simulate belt movement with bones.

### **Left Belt Wheels**

Here you set Wheel Colliders for the left belt, but only that is touching the ground. Plus for each, you select a bone that will have this wheel effect on.

### **Cosmetic Wheels**

These are wheels that are not on the ground. It's here to make them rotate.

## **Statistics**

### **Motor Torque**

This is torque in Newton-meters that will be applied on belts.

### **Brake Power**

This is torque in Newton-meters that will be applied as brake power on all axes with Brake boolean set to true.

### **Max Speed**

The maximum speed that a tank can achieve.

### **Turn Speed**

This is how fast the tank will turn in degrees per second.

### **Center Of Mass**

This sets the Rigidbody center of mass relative to the pivot point of the transform Rigidbody attached to.

## **Keys**

W - forward

S - back/brake

A - left

D - right

Space - handbrake

E - interaction

Right Mouse Button - stop turret and gun rotation

Left Mouse Button - Fire

# **Plane Controller**

## **Set up**

The first tab is Set up and here you can set up plane wheels, wings, guns, and other essential variables.

## **Wheels**

Here you can determine the number of wheels in a size field. For each wheel, you must select the wheel collider (Add wheel collider component to empty GameObject

and set the model of the wheel as the child of that GameObject to be visually rotating).

### **Back Turning Wheel**

If this boolean is set to true the turning wheel(s) must be at the back of the plane, when false the turning wheel(s) must be at the front of the plane.

For each wheel setup these booleans:

### **Wheel Collider**

Wheel collider which represents wheel functionality. A child of this object must be a visual model of the wheel.

### **Steering**

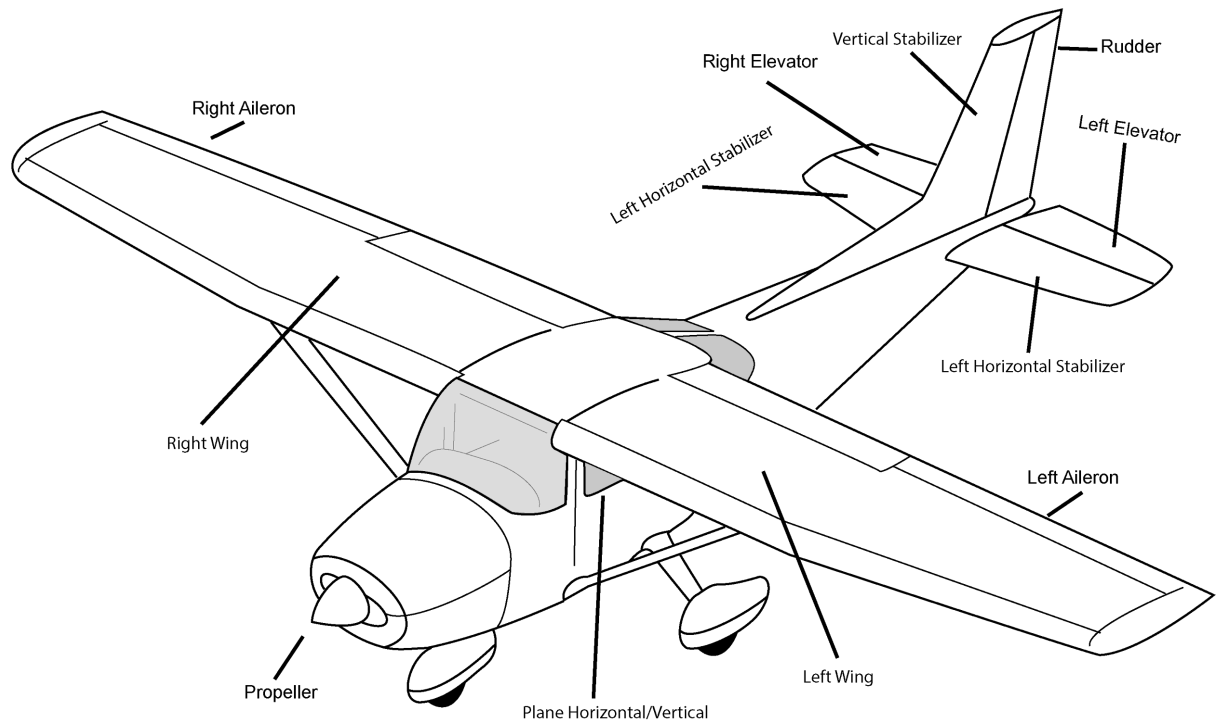
Turn this on if you want this wheel to be able to turn.

### **Brake**

Turn this on if you want this wheel to have brakes.

## **Wings**

Wings are physics-based. You need to set up their position and area in square meters.



### **Plane Horizontal**

Position should be the same as center of mass.

### **Plane Vertical**

Position should be the same as center of mass and should be rotated -90 in Z axe.

### **Rudder and Vertical Stabilizer**

Transformation must be rotated along Z axe -90.

## **Engine**

### **Propellers**

List of all Transforms (positions and models) of propellers.

## **Statistics**

In the second tab are variables that change statistics and characteristics of the plane.

**Engine Torque**

This is torque in Newton-meters that will be applied to properellers.

**Brake Power**

This is torque in Newton-meters that will be applied as brake power on all axes with Brake boolean set to true.

**Max Speed**

The maximum speed that a plane can achieve.

**Turn Speed**

This is how fast the tank will turn in degrees per second.

**Max Steering Angle**

Max steering angle of all wheels that have Steering boolean set to true.

**Steering Curve**

This curve determines how the Max Steering Angle is modified depending on the speed of the plane.

**Aerodynamic Curves****Center Of Mass**

This sets the Rigidbody center of mass relative to the pivot point of the transform Rigidbody attached to.

**Roll Range**

Define max and min angle of ailerons.

**Yaw Range**

Sets max and min angle of Rudder.

**Pitch Range**

Define max and min angle of Elevators.

# FAQ

They're not stupid questions, just stupid answers from us.

## **How does a user make a player controlled character from scratch?**

Add PlayerInput and SoldierMovementFromInputs scripts to the character.

## **How does a user make an AI controlled character from scratch?**

Add a SoldierMovementFromInputs script, and any script ending in “\_AI” such as Wander\_AI.

## **How does a user update the 3D prefab to use their own model?**

Simply replace the model prefab within the character. If you have scripts that reference bones, such as the Hands in the Soldier\_UsableHolder script, make sure you assign the new transforms.

## **How does a user update the weapon prefab to use their own model?**

Simply replace the model in the weapon prefab, then align the BarrelTip GameObject with where you want the bullet to come from.

## **How does a user update the vehicle prefab to use their own model?**

Create a Prefab Variant of the type you want, such as “Rideable\_Tank”, then put your own model in. Depending on the vehicle type, you may need to set up stuff on the root GameObject, such as the treads on the Tank, wheels on the Car, etc.

## **How does a user update the player UI to use their own UI?**

Various UI Scripts such as HealthBar can be easily added to custom UI elements.

They will usually automatically reference the relevant script in a parent GameObject, though some have the option to have a reference dragged in directly, such as the FillByFraction script. The easiest way to do this is to select the GameObject with the FillByFraction or other script on it, right click the other GameObject with the target script and select Properties, then drag the component into the FillByFraction's slot.

### **How does a user add their own animations and get them to work with types of Input.**

You can use an AnimatorOverrideController to swap out animations that you want in the included Soldier's Animator Controller.

### **How does a user swap the button needed for Player Input?**

Controls can be remapped on the PlayerInput script itself. The names of the inputs correspond to the entry in the Input Manager in Edit > Project Settings.