Full Game Kit: Hammer 2
Copyright 2015 Xform
Version 1.1.1

Thanks for buying Full Game Kit: Hammer 2!

Documentation
This (updated) document can be found here:
http://www.xform.nl/projects/AssetStore/Hammer2/Hammer2Documentation.pdf
But is also located inside the package Projects Assets folder!
This document is ‘work in progress’. Please check 'Version History Document' .

Support
Send your (support) questions to support@xformgames.com
There's also a thread on the Unity forum, although this is not an official page.

License notes
You may use all source codes and assets in your own projects and games, also if they are for commercial use. You may NOT distribute the source code or assets directly, for instance in the Asset Store. If you've obtained this software from any other source than the Asset Store,
your license is effectively invalid, and Xform cannot provide support for it.

Index

1 General Information

1. Installation

Download and import the package from the Unity Asset Store into a preferably new Unity Project. Alternatively you can try copying the already unpacked contents into a Unity Project.
If you're having trouble using your Xbox 360 controller, be sure to check if the ProjectSettings/InputManager.asset is updated correctly. If not, overwrite it with the InputManager.asset inside Hammer2InputManger.zip located in the Project Assets folder.

Updating from 1.0.x: Please see the FAQ.

2. Play the game in the editor

Open the StartUp (preferred), Loading or Game scene in the Scenes folder and press play.

3. Create a build

Create a new build:

a. Choose File > Build Settings.
b. Choose your platform and 'Switch Platform'.
c. Press Build and select a folder.

Supported and tested platforms:

1. Web Player
2. Standalone (Win, Mac, Linux)
3. WebGL (Preview)
4. Android [NEW] - See 4. Mobile Support

4. Controls

Keyboard

Control:	Action:
WASD/Arrows	Move
Mouse	Aim
Space/J	Jump or ascend or drift
LMB/Z	Fire
RMB/X	Grenade or descend
MouseWheel/Q/E	Change weapon
Shift/C	Activate HammerTime
Enter/E	Enter or exit vehicle
Esc/P	Pause
M	Mute audio toggle

Controller

Control:	Action:
Left Analog Stick	Move
Right Analog Stick	Aim
A	Jump or ascend or drift
Right Trigger/Shoulder/X	Fire
Left Trigger/Shoulder/B	Grenade or descend

D-Pad Left/Right	Change weapon
Left Stick Button	Activate HammerTime
Y	Enter or exit vehicle
Start	Pause

Touch

Control:	Action:
Drag left stick	Move
Drag right stick	Aim
Double tap right stick	Reset view
Tap up button	Jump
Tap blue button	Fire
Tap red button	Throw grenade
Hold timer button	Hammer Time!
Tap cycle button	Cycle weapon
Tap enter button	Enter vehicle
Tap pause button	Pause game

Cheats*

Key:	Cheat:
K	Mission Complete
L	Game Over
G	God mode toggle (no damage)
I	Toggle interface
O	Toggle player visibility
T	Almost get all achievements
U	Unlock missions
C	Give 100000 cash

***Not available on touch devices.**

5. Version history

1.1.1
- FIXED: Compatibility issue for XGradient.cs for Unity 5.2.2

1.1
- TIP: There is an issue in Unity 5.2 regarding alphabetical sorting. Turn this off in you hierarchy view!
- NEW: Android build support with mobile controls
- NEW: CursorManager (see CursorManager.cs)
- NEW: PoolManager (See PoolManager.cs, PoolData.cs and PoolVisualizer.cs)
- NEW: AutoAim (See Hammer.cs and CharacterManager.cs)
- NEW: Advertising (See AdvertisingSample.cs to see how you could show advertising)
- NEW: Mobile localization
- FIXED: Camera will look correctly when it has been offset by objects behind it.
- FIXED: Set camera layerCullDistances properly on camera creation
- FIXED: Some camera settings for turrets and tanks
- FIXED: Bunch of small non-critical issues.
- FIXED: Compatibility issue for XGradient.cs for Unity 5.2
- ISSUES: See Unity forums/issue tracker on Alphabetically sorting and particle position issues. There is a workaround for the last one in this package.

1.0.4

- FIXED: Vehicle collision NullReferenceException in Vehicle.cs

1.0.3
- FIXED: Localization typo
- FIXED: Animation Events being triggered with no component to handle it.
- FIXED: No more warnings when picking up weapons and gadgets while in a vehicle.
- FIXED: Removed some inactive vehicles from Game scene

1.0.2
- NEW: Added controls and cheats to the ReadMe
- FIXED: HammerTime controller issue
- FIXED: Animator no receiver warning and errors
- FIXED: AttackHelicopter controls where you would shoot and fly up at the same time.

1.0.1
- FIXED: Turret, MissileTurret, MammothTank and AttackHelicopter secondary fire controls.
- FIXED: F22 controller input.
- FIXED: Xbox 360 controller input not used properly.
- NEW: Added the InputManager.asset to the Project Assets folder for convenience.
- NEW: More comments in the UI related scripts.
- FIXED: Mammoth Tank explosion was missing a material reference.
- NEW: Added this document to the package.

1.0.0
- NEW: Initial build.

6. Version history document

1.1.1 November 4th 2015
- Added entry to FAQ on non-working iOS buttons.
- Added Chapter 6 - Mini-tutorials
- Added 2 tutorials.

1.1.0 September 16th 2015
- Added Android as supported build
- Changed FAQ to chapter 5
- Added Chapter 4 - Mobile Support
- Added FAQ entries.
- Added touch controls.

1.0.2 September 1st 2015
- Added controls from ReadMe.
- Updated version history.
- Fixed some typos.

1.0.1 August 30th 2015
- Updated version history of package.
- Added index.
- Added chapter structure and formatting.
- Added content for chapters.
- Added FAQ chapter.

1.0.0 August 21st 2015
- First!

2 The Game

1. Introduction

– A brief history of the game

"The city streets are packed with bad guys. Who you gonna call? That's right, it's THE HAMMER. He'll straighten them out with his shotgun."

Hammer 2 - Reloaded is the sequel to Xform's first game, The Hammer, released in 2004 as a 3d webgame built in Virtools. At the time, it was a groundbreaking 3d webgame inspired by triple A titles such as GTA3 and Max Payne.

Xform provided a tutorial series back in those days called Two Minutes Of Mayhem, which showed developers how to recreate the basic functionality of the game.

In 2013 Xform released Super Hammer, built in Director/Shockwave this time, which was their take on the infinite runner genre, featuring the iconic action hero.

Now, Hammer is back in an explosive way. And surprisingly, now in the form of this example project!

2. Instructions

– The goal of the game and how the controls work

In the game you control the fearless hitman The Hammer, and help him in his quest to cleanse the city of bad guys. The Hammer has 30 missions to complete and does this by destroying or killing the number of targets shown on the left side of the interface, and on the minimap.

The action is shown from a thirdperson perspective, and can be controlled by responsive aiming and running controls. The Hammer can shoot targets with his standard gun, or pickup different weapons and even grenades. He can also enter vehicles and turrets, which sometimes provide a lot more firepower. The Hammer also possesses a special power called Hammer Time, in which time slows down, and he can pickup special gadgets that help him or improve his weapons.

When the player has finished the game, he can try to complete the three challenges set for each mission. He can also try to get all the achievements, or to collect enough cash to buy all shop items. Some of which are not to be missed!

Note: For the game's control scheme, please check Controls in chapter 1.

3 Example project

1. General

– How to use this project and how to learn about it.

We've decided to sell this game on the asset store as an example project. To be honest, the main reason for this was that we thought we could make some money for all the time we had invested in creating it. We're hoping that there is a demand for more polished and complete example projects. Most of the projects out there are pretty basic and miss features to really publish them 'as is'.

Providing such a 'complete' project has the downside that you cannot make every aspect of it as modular and foolproof as some other simpler kits are. That's why we clearly state that it's for intermediate users!

Unfortunately, it's not do-able for us to describe every script in the project in a lot of detail here in the documentation. So we've created a brief (but necessary) overview of how the game does what it does.

Our first tip is to play the game for a bit. Stop the game and see what's going on in the hierarchy manager. Change something in the code and see where you've broken it. Start debugging. Most of the scripts have a summary what it does and a lot of coder comments.

And of course don't forget to read the rest of this document.

2. Flow

– Describing the coding flow, scenes.

The game starts in the StartUp scene. This is a perfect place to initialize some data before the player can do or see anything.

Then the game continues into the Loading scene, in which a loading bar is shown and the Resources are loaded in. So in fact, you cannot use stuff that has no reference in a scene before this scene is done!

After this, we enter the Game scene, in which the world is already there, and the Interface script takes over what happens next. This could be either to enter the menu, or to go the game directly.

Important note: Every time the player retries or enters a new mission or goes back to the menu, the whole game scene is reloaded, but with a different configuration.

When we're entering a mission, the LevelScript and MissionManager take it from there, mainly enabling and configuring objects in the scene, and instantiating things from the resources.

3. Data Management

– How important variables are set and used.

The engine and game variables are initially defined in text files that are read in at runtime in the editor, and are included in the build for deployment. This way, it's easy for an artist or designer to tweak and configure stuff without the hassle of editing scripts.

Globals.txt contains important game configuration variables such as resolution and cheats.
Inside the game, variables are managed through Data.cs (engine variables) and GameData.cs (game specific related variables). Some of the variables are directly tied to the .txt files, but you can also add your own variables that cannot be initially set through Globals.txt, but can be easily accessed throughout the game.

Important: If you want to save specific variables using UserData.Save(), be sure to add the variables in the Globals.txt SaveList variable.

SharedData.txt contains more specific game data, such as object and mission configurations. These are accessible in the form of dictionaries, and are only used for reading in data!

You can easily add your own data sets in SharedData.txt. Be sure to obey the formatting rules. Also, you can freely add as many parameters to the dictionary as you want. Keep in mind that it remains manageable and readable.

Localization text files such as **Resources/Localization/English (US).txt** contains only textual / editorial information, also read only. Of course, only the currently selected language and corresponding text file can be accessed.

You can easily add your own entries. Make sure the key is unique. And that you add it to the other languages as well.

4. Code

– Describing some of the game's important systems.

As you might have seen, the scripts are divided in to two main categories and folder:

Engine: All scripts that we want to share between the different games we produce.
Game: Specific code for the Hammer 2.
And then there's a third called **XGUI**: Our quick solution for handling stuff previously done by awesome third party software.

We hope to have some more global information on the systems in the future versions of the document. For now, please refer to the comments in the scripts themselves.

5. Assets

– How the resources are structured, and how they are used.

All of the assets below are included in the build in one form or another. Some of them are already present in the game scene (i.e. buildings, and also vehicles), others exist only in the Resources. These Resources are loaded together with scene/level 3 (= the game scene). Be sure to check the Build and Player Settings for this.

Adding Assets: For the 3d assets, it's important to understand that in this project, the dependencies of a prefab inside the Resources/Prefab can be located in the Resources/3D, Resources/Materials, Resources/Textures and Resources/Animations. We advise you to obey to this formatting when adding your own assets.

Note: Most assets that are used in the game are processed by a manager when the game starts. Also check the SharedData.txt for examples.

Exporting Assets: If you want to use particular prefabs in a different project, we advise you use the 'Export Package' functionality by right clicking on the object in the Project tab.

Lowpoly characters and animations
These are located in Resources/Prefabs/Character. These are all instantiated in the CharacterManager.

Lowpoly vehicles, buildings, streets, misc.
These are located in Resources/Prefabs/ folder as well. Most of these are already present in the game scene and are processed when a mission starts by either the DestructibleManager and/or the VehicleManager.

Lots of effects, flashes, particles and explosions
These particles are all game objects inside prefabs that are located in Prefabs/Effects. For examples of their use, please see the managers inside the Effects scripts folder.

5 royalty free music tracks for you to use
These are located in the Audio/SFX. These are played using the AudioManager functions.

Over a 100 crystal clear game sound effects
These are located in the Audio/SFX folder. These are played using the AudioManager functions. There's a difference between simpler 2d interface sounds and 3d sounds, which can be configured in more detail.

Loads of icons, buttons and backgrounds
These are all located on Atlasses, which can be found in the Sprites folder. Most of these are already assigned to Image components in the Interface hierarchy. If sprites need to be set directly in code, this is done through the AtlasManager script.

4. Mobile Support

As of version 1.1 Android support has been implemented and tested.
https://play.google.com/store/apps/details?id=com.xformgames.hammer2

Keep in mind that you need the Android SDK installed!
See: http://docs.unity3d.com/Manual/android-sdksetup.html

Building an Android version :

a. Import package into a new project or update your existing project
   (Check for issues if you already changed stuff! A lot has been changed!)
b. Click File > Build Settings
c. Select "Android" and click 'Switch Platform'
d. Open Globals.txt
e. Change 'Platform' to 'Android' (OR iOS if you fancy a shot) **THIS IS THE MOST IMPORTANT STEP!**
f. Change 'Branding' to 'XformMobile'
   (This is optional, but there is mobile specific localization. For example text in the HelpPanel)
g. Change 'Quality' to 'Fastest'
   (We do not recommended keeping it 'Fantastic'. There are lots of post-processing effects AND there might be skybox issues. Furthermore, we need speed and performance)

h. Press play and take a look at the game - see, play and mess around

   Optional: You might get better performance by switching shaders. Although we had different results Choose Assets/Hammer 2 - Quick Shader Change/Standard To Mobile Diffuse Shader in the top editor menu. This will change all Standard materials to a default Mobile/Diffuse shader - You can revert this by choosing Assets/Hammer 2 - Quick Shader Change/Mobile Diffuse To Standard Shader
   Make sure you save your project after this!

i. Click File > Build Settings
j. Click 'Player Settings'

k. Check 'Resolution and Presentation' and set your preferences if necessary
l. Check 'Icon' and set your preferences if necessary
m. Check 'Splash Image' and set your preferences if necessary
n. Check 'Other Settings' and set your preferences. See image on next page with our settings
o. Check 'Publishing Settings' and setup your keystore if necessary

p. Click 'Build' or 'Build and Run' if you have a device connected - Select folder and continue

Differences in the Android platform version:

Touch Controls: Most of the layout is still the same and we added some additional buttons. Including 2 joysticks. The joystick code is based upon joystick.cs available in the CrossplatformInput, but is has been seriously modified so it is available as MobileJoystick.cs in the root of Scripts/Game.

Performance: For performance reasons we use a lot of WebGL settings for this build as well. For example a 'Low Quality' camera, maximum allowed characters (at the same time), forward rendering, etc.

Optimizations: Furthermore, there is a new PoolManager so we can re-use lots of objects that would otherwise be created/deleted all the time.

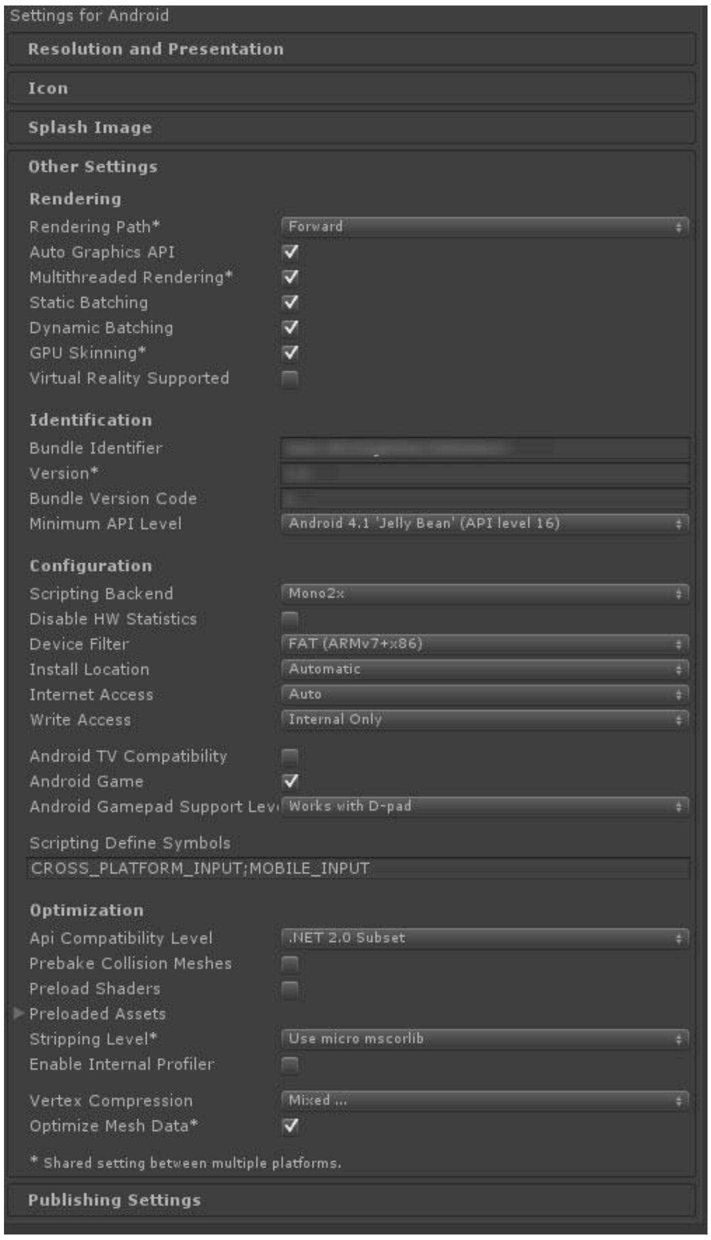
Auto Aim: Finally, there is some autoAim functionality to make killing enemies easier. It can be hard to have a perfect aim with touch controls so here we help you and the end-user out a bit!

Comments: All new code is commented!

Android Player Settings: See the image below.

Notes: If you are getting CrossPlatformInput errors in your console make sure you are in the correct platform AND that you have set the 'Platform' variable in Globals.txt properly. CrossPlatformInput does not work (properly) as a StandAlone build!

Note: Please search for [NEW], [MOBILE], [ANDROID] and [BUG] (in one case) to see and read about the latest changes.



Settings for Android

Resolution and Presentation

Icon

Splash Image

Other Settings

Rendering

Rendering Path*	Forward
Auto Graphics API	✓
Multithreaded Rendering*	✓
Static Batching	✓
Dynamic Batching	✓
GPU Skinning*	✓
Virtual Reality Supported	

Identification

Bundle Identifier	[redacted]
Version*	[redacted]
Bundle Version Code	[redacted]
Minimum API Level	Android 4.1 'Jelly Bean' (API level 16)

Configuration

Scripting Backend	Mono2x
Disable HW Statistics	
Device Filter	FAT (ARMv7+x86)
Install Location	Automatic
Internet Access	Auto
Write Access	Internal Only
Android TV Compatibility	
Android Game	✓
Android Gamepad Support Lev	Works with D-pad

Scripting Define Symbols

CROSS_PLATFORM_INPUT;MOBILE_INPUT

Optimization

Api Compatibility Level	.NET 2.0 Subset
Prebake Collision Meshes	
Preload Shaders	
Preloaded Assets	
Stripping Level*	Use micro mscorlib
Enable Internal Profiler	
Vertex Compression	Mixed ...
Optimize Mesh Data*	✓

* Shared setting between multiple platforms.

Publishing Settings

5. FAQ

1. What is this?
It's a full game AND a Unity example project. Play the game and have fun! Explore the example project and get inspired. Use the scripts and assets in this project for your own work as you see fit.

2. Why isn't feature x included?
The project started out as just a game project, so it only has the functionality we thought was necessary for it. The game already has quite a bit of functionality, maybe even more than it should have for simplicity's sake. Nevertheless, if you think something essential is missing, don't hesitate to let us know, and also why you think it should be included.

3. Can I skip directly into the game?
Yes. In Globals.txt, set the Scene variable to Level instead of Menu. You can also skip the Preloader and Splash by setting these to false. Note: A Webplayer build needs the Preloader to be true.

4. What is the easiest way to upgrade to a newer version of the kit?
There is no real easy way unfortunately. Try to keep your own programming outside the FGK Hammer 2 scripts. Make a backup of your work. Import the upgraded package into a new project. Close Unity and Monodevelop. Copy the freshly imported files over the ones already there, making sure you're not overwriting your own work.
If you are already using 1.0.x version of the package, we really recommend importing your own scripts and assets into a fresh project that has the new package imported.

5. Does the game work on platform x?
There's only one way to find out: go for it. Please know that it does not help to bully us into adding features or platform support by giving us bad reviews or ratings. Obvious really.

6. Can I sell games made with this kit or based on this project?
Yes. You can even sell it if you've just made a mod or a cool adjustment to the original game. Don't forget to buy us a beer if you become a billionaire.
Please do not blame us if you get accused of theft or asset flipping by the community.
Please do not use the 'Hammer 2 - Reloaded ' or Xform related names, logos, videos or screenshots in your publications.

7. I found a bug. Can I report it?
Yes. We hope you have time to investigate it first yourself thought, and maybe suggest a solution if you have one. Send it to the support mail address.

8. I want to change game parameters and settings. Where should I start?
Read the bit about Data Management in this document.

9. I want to add some of my own assets. Where should I start?
Read the bit about Assets!

10. When is the next update planned?
We're still looking at optimizations, improvements and bug fixes on the current build. If we make a critical change, we will update the package as we can get it through the asset store review process!

11. Why is this project so cheap?

Mostly because the systems and scripts are not as modular and easily portable as you sometimes see in other Asset Store packages. This may change in the future.

12. How can I use the cheats?
You can enter 'cheat' + enter in the menu (you will hear a soundeffect when you've done it correctly), or you can change the variable Cheats in Globals.txt to 'true'.

13. Is the package compatible with Unity 4.x ?
We just cannot get it to work in 4.6.7. The project was made in Unity 5 and upgrading (downgrading) gives some bad editor errors. We cannot see the objects in the scenes and prefabs cannot be added. We started fixing the other errors we anticipated, but it still does not work. In case you've accidentally bought the package: you can always install a Personal Edition of Unity 5.1.2+ to test and run the project, or to get access to the assets.

14. Is the package compatible with Android?
Yes. From version 1.1 and up we have implemented support for Android devices.

15. The game doesn't run or runs poorly on my Android device x. Will you fix this?
We don't have a lot of Android devices. The main development device was a Samsung S3 Galaxy. Performance on this device is good. The game can be properly played. If you want it to run (perfectly) on even older devices you have to start optimizing more. Although we cannot give guarantees on all devices in the world, we do appreciate it if you provide us with feedback on the support mail address. As long as it is not abusive.

16. Is the package compatible with iOS now too?
There is no reason to believe this version could not work for iOS! But also see FAQ question 5 and 15.

17. Some buttons aren't working on iOS?
Try a different Stripping Level or disable it altogether. Some necessary things are probably stripped otherwise.

6. Mini-tutorials

0. Basic information for all tutorials

First open up Globals.txt and set Scene to Level, LoadUserData to false and Mission to 1. This is a setup we always use to quickly start in a mission to test new stuff. Optionally you can set SkipGetReady to true. This will start your mission even faster.

If you want to try/develop something vehicle related change the Mission to 99. Mission 99 is a vehicle test environment. In Globals.txt set both BuildAI and AIController to false. If you want to test/develop something AI related then keep this true ;)

Don't forget to revert these settings when making your (final) build!

1. Creating dynamic objects you can drive into.

For this mini-tutorial we are going to make a TrafficLightCornerPole static, but dynamic when you drive into it with a vehicle. You can best test/develop this in mission 1. So read **0. Basic information for all tutorials** first.

Open up SharedData.txt and scroll to line 628. Here you will see how dynamic destructible objects are setup. Pay specific attention to the dynamic bool and mass.
Now go to line 649 and add *#dynamic: true, #mass: 10* to TrafficLightCornerPole. Check for typos!

Your entry should look like:

```
TrafficLightCornerPole        [#health:  5.0, #score:  10, #impact: RicochetMetal, #explosion: DefaultSmall, #dynamic: true,
#mass: 10, #lifetime: 0.0]
```

If you start the game you will see both cornerpoles in Mission 1 fall over. Now we are going to freeze them.

First open up DestructibleData.cs as we're going to add some specific additional destructible data. Under 'public bool dynamic' add 'public bool freeze = false;'. Save the file.

Return to line 649 in SharedData.txt and add #freeze: true, after #dynamic: true;. Check for typos!

Your entry should look like:

```
TrafficLightCornerPole        [#health:  5.0, #score:  10, #impact: RicochetMetal, #explosion: DefaultSmall, #dynamic: true,
#freeze: true, #mass: 10, #lifetime: 0.0]
```

Please note the order of the entries are irrelevant, but I like to keep them in order of the DestructibleData.cs as much as possible. Again check for typos. It is important that the names of the entries are the same as the names of your variables in your DestructibleData.cs.

If you start Mission 1 nothing is different, but the DestructibleData for the TrafficLightCornerPole should be updated with freeze is true.

Now open up Destructible.cs. In line 23 you will see we add a RigidBody with mass if the destructible object is dynamic. We will change something here.
Change the current code:

```
if(destructibleData.dynamic) gameObject.AddComponent<Rigidbody>().mass = destructibleData.mass; // dynamic
else { // if not dynamic (those objects are never parented) and we should reparent; get the parent below and parent it
	if(destructibleData.reParent) {
		Transform newParent = GenericFunctionsScript.GetParentBelow(gameObject.transform.position);
		if (newParent != null) gameObject.transform.parent = newParent;
	}
}
```

into:

```
if(destructibleData.dynamic) {
	gameObject.AddComponent<Rigidbody>().mass = destructibleData.mass; // dynamic
	if (destructibleData.freeze) gameObject.GetComponent<Rigidbody>().constraints=RigidbodyConstraints.FreezeAll;
} else { // if not dynamic (those objects are never parented) and we should reparent; get the parent below and parent it
	if(destructibleData.reParent) {
		Transform newParent = GenericFunctionsScript.GetParentBelow(gameObject.transform.position);
		if (newParent != null) gameObject.transform.parent = newParent;
	}
}
```

If you start Mission 1 the cornerpoles won't fall over, but nothing will happen if you drive into them either.

No go to line 48 (below the Damage() function) and add an OnCollisionEnter function. Like so:

```
void OnCollisionEnter(Collision collision)
{
	if (collision.collider.gameObject.layer == 15){ // Only with an object from the vehicle layer!
		Rigidbody rigidBody = gameObject.GetComponent<Rigidbody>(); // Check for RigidBody
		if (rigidBody != null && destructibleData.freeze) rigidBody.constraints = RigidbodyConstraints.None;
	}
}
```

That is all! Start Mission 1, walk right, enter the tank and drive over the TrafficLightCornerPole! Watch out for the FireHydrant just in front of it. It isn't dynamic and/or freezed so you will drive into it!

2. Cloning an existing enemy (and some spawner info)

For this mini-tutorial we are going to add another enemy. Which will be a clone of EnemySuit. You can best test/develop this in mission 1. So read **0. Basic information for all tutorials** first.
This tutorial will also teach you how to adjust/change spawners.

First of all we will change some spawners in Mission 1 to spawn our new (non-existing) enemies.
Open up the Game scene.
Select and open the Mission object in the Hierarchy tab.
Open Mission 1, then Enemies and select all EnemySuitSpawner objects (4).

These are spawners located in front of 4 buildings (2 left, 2 right) in the main street.
In the Inspector change the Spawner entry to MediumSuit2, then save your scene and optionally your project.

Open SharedData.txt and go to line 230. Here you have the entry for the MediumSuit spawner.
Clone this line and rename it to MediumSuit2.

Your entry should look like this:

```
MediumSuit2			[#type: Enemy, #subType:	EnemySuit, #interval: 0.5, #amount: 10, #unlimitedAmount: false,
#range:100.0, #minMaxOffset: [5.0, 15.0], #rotation: Offset]
```

Now start the game. You can start from the Game scene, but make sure you revert to StartUp when doing your (final) build. In-game nothing is changed, except some names, but everything should work.

Go back to SharedData.txt and in line 231 (your new MediumSuit2 spawner) change #subType to 'EnemySuit2'. This will be a new character to be spawned.
Now go to line 284 and clone the EnemySuit line.
Rename it to EnemySuit2. You guessed it, the same as the subType ;)

If you start the game nothing happens! Thankfully nothing crashes either! :) What happens is that the spawner objects are trying to spawn enemies which are not 'registered' yet.

Open CharacterManager.cs and go to line 52.
After this line (case "EnemySuit":) add another case; case "EnemySuit2":
Now start again and see the same enemies being spawned!

So we are nearly there. They only thing left is to actually create a new character prefab.
Go Back to line 285 in SharedData.txt(your EnemySuit2 entry) and change the prefab entry to EnemySuit2

Now we need to create a new prefab named EnemySuit2_Prefab.
In your project overview go to */Resources/Prefabs/Enemies/*
Right click and create a new prefab named *EnemySuit2_Prefab*.
Now click and drag the EnemySuit_Prefab (the original) onto the new one.
Save your scene and project.

The only thing left is to change the visual appearance of the prefab.
Drag the new prefab in your scene and do what you like.
I would create a new material in */Resources/Materials/Enemies/* called *EnemySuit2_Material*.
Select the EnemySuit model in your Scene and assign your new material in the Inspector.
Click Apply in the Inspector to commit your changes. Save your scene and your project.

If you start the game you will notice errors in Weapon.cs. Things go wrong here because we're using EnemySuit2 as a name. Most of the code is looking for and using (part of) the prefab name.
A lot of bodyparts of the enemy are still called EnemySuit, so things are bound to go wrong. Now comes the final and hard part. We will solve this with a little 'hack'!

Open up Enemy.cs and go to line 23; the function InitializeSpecific().

Anywhere in this function add:

```
if (characterData.prefab == "EnemySuit2") characterData.prefab = "EnemySuit";
```

Optionally can have some more fancy code that could check for an int at the end of this string and remove that int. So you can safely add EnemySuit3, EnemySuit4, EnemySuit5, etc. without the need for adding more ifs or a switch.

What we do is a bit cheating but at least it will give us new enemies quickly.

We are pretty much there.
If you start the game you will see you new enemies but things will still error when they start shooting at you.This is a bit of an oversight on our end.

At the end of the InitializeSpecific() function you will see we're looking for a *spine* object. We're creating a string based upon *type + " Spine1"*, but this should be characterData.prefab as well.

So quickly change this to

```
spine = GenericFunctionsScript.FindChild(gameObject, characterData.prefab + " Spine1");
```

That is it! Start the game and see your new visual enemy! Make an awesome new texture and continue from there!