

INTRODUCTION TO MODDING

FOR OPERATION FLASHPOINT / ARMA: COLD WAR ASSAULT

Introduction

„Modding“ (in the context of video games) refers to modifying a game by its players. Original content can be changed or new one can be added like replacing textures/models, tweaking speed/damage/armor values or adding new maps. It's done by fans of the game (as opposed to the original developers) for the purpose of making the game more entertaining.

OFP is relatively easy to edit and so the users started to mod for it, even though it wasn't the intention of Bohemia Interactive when they were developing the game. Later they released official modding tools and the modding scene grew further. It peaked in years 2005-2007. Since then, most of the people have moved to ARMA games. Today there are few active users and sometimes somebody new comes and asks newbie questions. This guide is intended for the latter.

Modding for OFP is both easy and hard. It's easy because you can pull something together relatively quickly, there are many different areas you can change and it's not as complicated as in later ARMA games. On the other hand, it's hard because it was never documented properly, testing is not very convenient (you'll have to restart the game a lot) and errors are rather vague.

This document consists of two parts: one is a basic tutorial to quickly make a simplest modification. Second part is a mini-encyclopedia of different file types present in the game. I'm not going to cover every single aspect of the game. Instead I'll point out other tutorials where you can find more information.

I'll be using 1.99 version of the game which you can find on Steam and GOG:

- https://store.steampowered.com/app/65790/ARMA_Cold_War_Assault/
- https://www.gog.com/en/game/arma_cold_war_assault

Part I – Basic Modding Tutorial

1. Modifying the main config

Reading and modifying configuration files is essential to OFP modding. In this chapter I'll show you how to edit the main config. You'll need some tools so follow these steps:

- Download [Notepad++](#) and install it
- Download [Notepad++ syntax highlighting for OFP](#) and extract it
- Launch Notepad++ and from the top menu select Language → User Defined Language → Define your language... → Import...
- Import each of the four XML files from the extracted archive and then restart Notepad++
- Download [Converted configs](#) and extract it
- Copy `bin\config199.cpp` to the BIN folder in your game directory
- Rename the file to `config.cpp`
- Open it in Notepad++
- From the top menu select Language → EXT C-Like
- From the top menu select View → Fold all (or ALT+0 on the keyboard)

Now you should have something like this:

```
//class CONFIG.cpp {
+class CfgExperience {
+class CfgTextureToMaterial {
+class CfgMaterials {
+class CfgModels {
+class CfgVehicleActions {
+class CfgManActions {
+class CfgMovesMC {
+class CfgAmmo {
+class CfgRecoils {
+class CfgWeapons {
+class CfgCloudlets {
+class WeaponFireGun {
+class WeaponFireMGun: WeaponFireGun {
+class WeaponCloudsGun {
+class WeaponCloudsMGun: WeaponCloudsGun {
+class CfgCloth {
+class CfgVehicles {
+class CfgCrew {
+class CfgNonAIVehicles {
+class CfgSurfaces {
+class CfgFonts {
+class CfgWrapperUI {
+class CfgInGameUI {
+class CfgDetectors {
+class CfgGlasses {
+class CfgFaces {
+class CfgFaceWounds {
+class CfgMimics {
+class CfgVoice {
+class CfgHQIdentities {
+class CfgSFX {
+class CfgEnvSounds {
+class CfgDestroy {
+class CfgHeads {
+class CfgEffects {
+class CfgWindows {
+class CfgMusic {
+class CfgSounds {
+class CfgTitles {
+class CfgIntro {
+class CfgCredits {
+class CfgCutScenes {
+class CfgCameraEffects {
+class CfgMarkers {
+class CfgMarkerColors {
+class CfgMarkerBrushes {
+class CfgWorlds {
+class CfgWorldList {
+class CfgGroups {
+class CfgAddons {
//};
```

This file is a database that determines various aspect of the game. For example: `CfgAmmo` defines what kind of bullets the game will have, `CfgWeapons` – guns, `CfgVehicles` – objects, `CfgMusic` – soundtrack etc.

Classes are containers that store properties and their values. Here's a mock example – a soldier with some armor and a name:

```
class soldier {
    armor = 3;
    displayName = "Barry";
}
```

It starts with a keyword "class" followed by name of the container. Brackets determine where its contents start and end. Between brackets there may be multiple properties. Each property has a name (e.g. "armor") followed by equality sign and a value (e.g. "3"). Value can be a number or a text in quotation marks or something else. It depends on the type of property.

Properties are separated from each other by a semi-colon or a new line (or both). I could also write the above example in this way:

```
class soldier {armor=3;displayName="Barry";}
```

It won't make a difference for the game. However, keeping one property per line is more readable.

A class may contain other classes:

```
class vehicles {
    class soldier {
        armor=3;
    }
    class tank {
        armor=400;
    }
}
```

Here `vehicles` includes a `soldier` and a `tank` which have different armor values.

Classes may copy properties from each other (which is also called "inheritance"). It's done by adding a colon and a name of the source class before the opening bracket.

```
class vehicles {
    class tank {
        armor=400;
        hasGunner = 1;
    }
    class m1abrams : tank {
        armor=900;
    }
    class m60 : tank {
        armor=300;
    }
}
```


In the above example `m60` and `m1abrams` are copying properties from the previous class `tank` so that they both have `hasGunner=1` (I don't have to type it manually into every tank). They also introduce their own `armor` which overrides the original `armor=400` value.

Real game config works in a similar manner. There's a class `CfgVehicles` which contains a list of all objects available in the game (e.g. buildings, soldiers, vehicles). In Notepad++ click on the plus icon beside it to make its contents visible.

```
8377 class CfgVehicles {
8378     access = 1;
8379     vehicleClass[] = {"Men","C.
8380     class All {
8571     class Logic: All {
8582     class AllVehicles: All {
8585     class Land: AllVehicles {
8597     class LandVehicle: Land {
8624     class Car: LandVehicle {
```

Scroll down until you find `class SoldierWB` and then click on its "plus" to unfold it. This class defines the standard US army soldier.

```
class SoldierWB: Soldier {
    model = "MC vojakW2";
    hiddenSelections[] = {"medic"};
    moves = "CfgMovesMC";
    vehicleClass = "Men";
    scope = 2;
    side = 1;
    accuracy = 0.7;
    displayName = "$STR_DN_SOLDIER";
    weapons[] = {"M16","Throw","Put"};
    magazines[] = {"M16","M16","M16","M16"};
    cost = 40000;
};
```



In the first line you can see that it copies properties from `Soldier`. If you look up the latter you'll find that it copies from some other class called `Man`. In fact, the chain of the dependencies looks like this:

`SoldierWB` → `Soldier` → `Man` → `Land` → `AllVehicles` → `All`

Not all of the sub-classes in the `CfgVehicles` are usable in the game. Some of them (like `Soldier`, `Man`, `Land`, `AllVehicles`, `All`) are only used as templates for other classes. Here `class Soldier` serves as a basis for playable soldiers. Class `All` contains all possible properties for objects in the game with their default values. Some types of objects use only certain properties (for example `hasGunner` is meant for vehicles and not for soldiers).

Property `scope` determines whether the class will be accessible in the Mission Editor (value 2 makes it visible and 0 hides it).

Now change the values of the following properties in the class `SoldierWB`:

```
displayName = "TEST";  
weapons[] = {"G36a"};  
magazines[] = {"G36aMag"};
```

Save the file and launch the game. Open Mission Editor and insert this soldier. It should look like this:

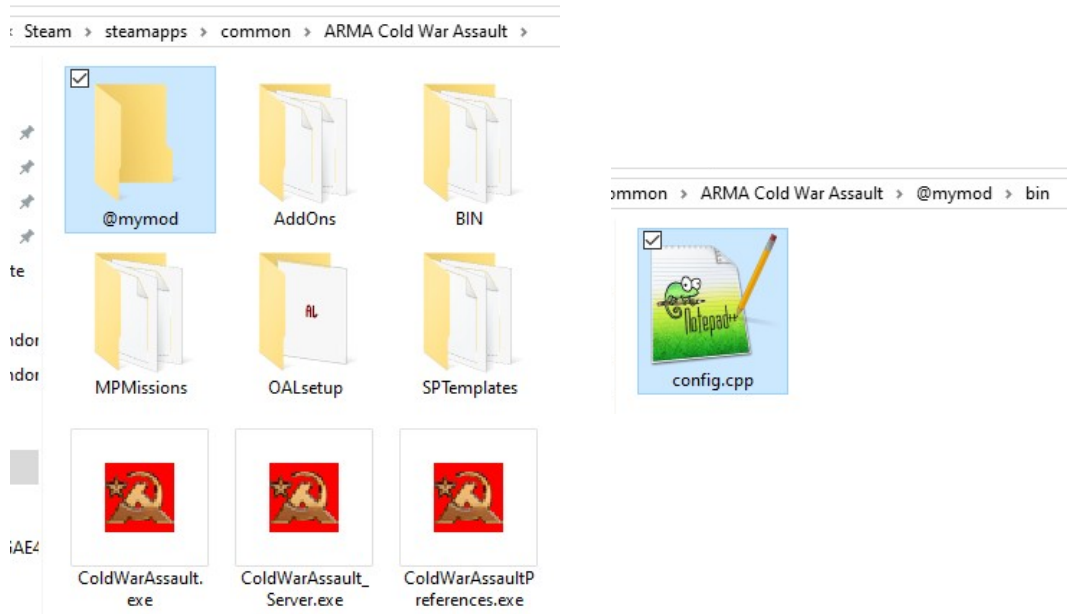


Congratulations! You've just modded the game. Name has been changed from "Soldier" to "TEST" and his gun from M16 to G36.

2. Creating a modfolder

It's best to separate your modification from the original game. This is what modfolders are for. Open the game directory, create a new folder and name it `@mymod`. Inside create another folder and name it `bin`. Then move your `config.cpp` there. It should look like this:

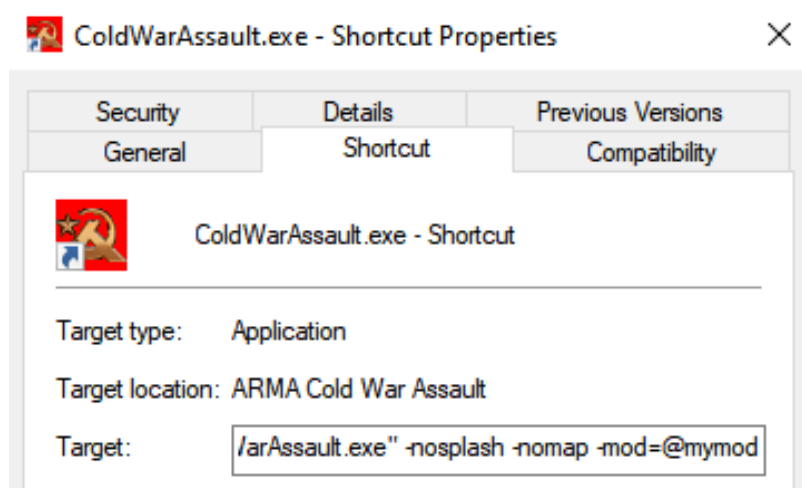
It's best to separate your modification from the original game. Go to the game directory, create a new folder, name it `@mymod`. Inside create another folder and name it `bin`. Now move the `config.cpp` that you've just edited into this folder.



To play with the modified config launch the game executable with a parameter `-mod=` like this:

```
ColdWarAssault.exe -mod=@mymod
```

It can be done via shortcut. Right-click on the `ColdWarAssault.exe` and select "Create Shortcut". Then right-click on the shortcut and select "Properties". Write parameters in the "Target" field:

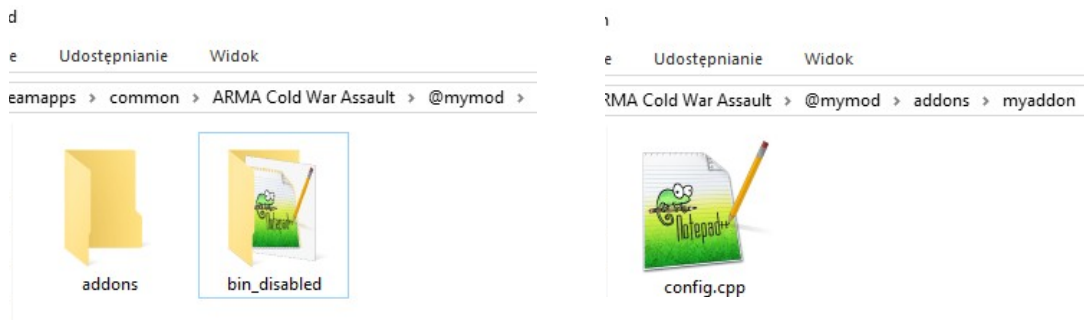


Now if you launch the game with that shortcut you should have default soldier with G36. If you launch the game normally it should be default M16.

3. Creating an addon

Addons are extensions that can be easily added without modifying the original game files (such as `bin\config.cpp`). It's more convenient to distribute (it's a single file), navigate (only including relevant classes in the config instead of hundreds of them) and combine with other modifications (you're not replacing content but adding new).

Go to the `@mymod` directory. Rename `bin` folder to something else to disable it (e.g. `bin_disabled`). Now create a new folder and name it `addons`. Inside create another folder and name it `myaddon`. There create a new text file and name it `config.cpp`. Open it in Notepad++.



It's going to work similarly to the main game config. It can include all the same classes as in the main game config (like `CfgVehicles`) except that it's not going to replace them but add new content.

First you need to add a special class called `CfgPatches` that will contain a signature of your addon.

```
class CfgPatches {
    class myaddon {
        units[]={};
    };
}
```

Next add `CfgVehicles` with a sub-class copying from the `SoldierWB` with properties that you've used before:

```
class CfgVehicles {
    class mysoldier : SoldierWB {
        displayName = "TEST";
        weapons[] = {"G36a"};
        magazines[] = {"G36aMag"};
    }
}
```

This will actually not work yet because the game needs the `SoldierWB` class to exist in this config. Do it this way:

```

class All {};
class AllVehicles: All {};
class Land: AllVehicles {};
class Man: Land {};
class Soldier: Man {};
class SoldierWB: Soldier {};

class mysoldier : SoldierWB {
    displayName = "TEST";
    weapons[] = {"G36a"};
    magazines[] = {"G36aMag"};
}

```

Now we're copying these classes from the main config. They are secured so that you can't modify them from an addon config. All you can do is copy.

Complete config should look like this:

```

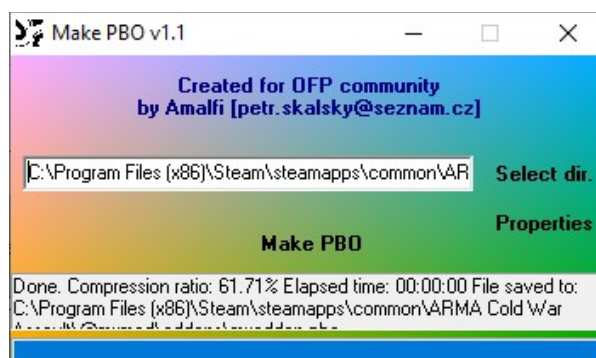
class CfgPatches {
    class myaddon {
        units[]={};
    };
}

class CfgVehicles {
    class All {};
    class AllVehicles: All {};
    class Land: AllVehicles {};
    class Man: Land {};
    class Soldier: Man {};
    class SoldierWB: Soldier {};

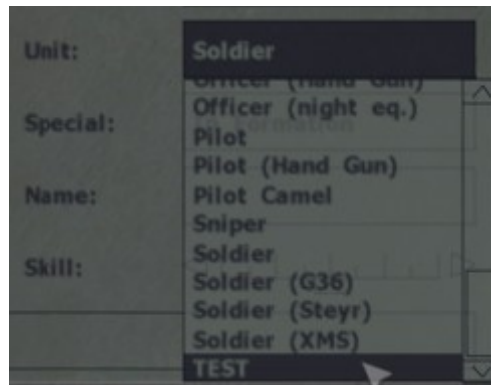
    class mysoldier : SoldierWB {
        displayName = "TEST";
        weapons[] = {"G36a"};
        magazines[] = {"G36aMag"};
    }
}

```

Download program [MakePBO](#) 1.1 by Amalfi. Extract the archive and run MakePBO.exe. Click on "Select dir" button and select "myaddon" folder. Then click on the "Make PBO" button.



Finally, launch the game with the shortcut you've made before. Both "TEST" and "Soldier" should be available in the Mission Editor. We've added a new unit called "test" instead of replacing the standard soldier.



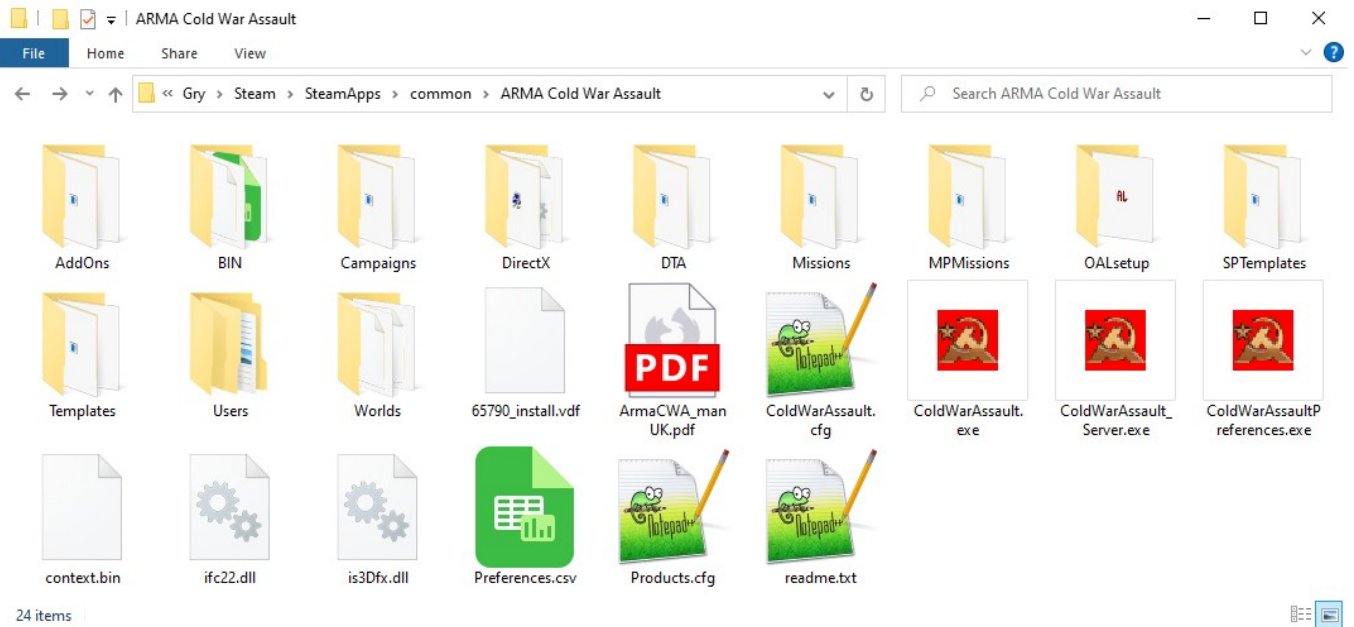
You can also drop `myaddon.pbo` in the `ARMA Cold War Assault\Addons` folder if you want to use it without a modfolder.

Congratulations! You've made your first addon. Feel free to play around with other properties (try making a soldier with an increased armor).

Part II – Game Files

1. Directory Structure

Let's look at the folders that come with the 1.99 version of the game:



AddOns

Stores addon files.

BIN

Stores game configuration files.

Campaigns

Stores campaign (a series of connected single-player missions) files.

DTA

Stores assets (e.g. textures, models, animations, sounds).

Missions

Stores single-player scenarios.

MPMissions

Stores multiplayer scenarios.

SPTemplates

Stores single-player scenario templates for the Mission Wizard.

Templates

Stores multiplayer scenario templates for the Mission Wizard.

Users

Stores player data (e.g. configuration, saves).

Worlds

Stores 3D maps of the original four islands.

2. BIN folder

There are four files inside:

CONFIG.BIN	Determines various aspect of the gameplay (e.g. objects, animations, sounds).
ijl15.dll	Library for loading jpg files. Not relevant for modding.
RESOURCE.BIN	Determines how the user interface looks.
STRINGTABLE.CSV	Contains game text in multiple languages.

You can find extracted "config" and "resource" in the `configs_extracted.zip` package. If you put `config.cpp` or `resource.cpp` in this folder then the game will load them instead of BIN files.

3. Dta folder

This folder contains multiple PBO with various game assets:

Abel.pbo	Malden island textures
Anim.pbo	Soldier animations
anim.s.pbo	Cutscenes shown in the main menu
Cain.pbo	Kolgujev island textures
Data.pbo	Textures

Data3D.pbo	3D models
DTAEXT.PBO	Inventory icons and information
Eden.pbo	Everon island textures
Fonts.pbo	Fonts
LandText.pbo	Ground textures
Merged.pbo	Miscellaneous textures
Misc.pbo	Miscellaneous textures
Music.pbo	Soundtrack
scripts.pbo	Various scripts used in the game
Sound.pbo	Sounds
Voice.pbo	Radio voices

4. File types

Here's a list of different file types and how the game uses them:

cpp	Configuration files (config.cpp and resource.cpp) are used to determine various aspects of the game (like available soldiers).
csv	Stringtable.csv provides different language versions
ext	Description.ext used in campaigns or missions contains metadata about campaign/mission
fxy	Font information. Determines the location of a given character in the font texture
html	Formatted text. briefing.html is mission briefing. Overview.html is displayed when you select the mission and equipment.html is displayed in the inventory section
jpg	Image file
lip	Used along with sound files to determine how soldier's lips should move
ogg	Compressed sound file
p3d	3D model
paa	Texture file
pac	Same as paa
pbo	Archive that stores other files. Similar to ZIP or RAR but meant specifically for OFP
rtm	Animation file
sqf	Similar to sqs but with a slightly different syntax
sqm	Core mission file. Stores all the things you've inserted through the Mission Editor
sqs	Code script. Used to in addons and missions to make create cool effects and events

wav	Uncompressed sound file
wrp	Island file. Determines island shape and objects on it
wss	Compressed sound file

CPP, CSV, EXT, FXY, HTML, LIP, SQF, SQM, SQS are files you can view in Notepad++. Others require special tools.

5. Q&A

What kind of options description.ext has?

<https://www.ofpec.com/editors-depot/index.php?action=details&id=26&game=OFP>

<https://community.bistudio.com/wiki/Description.ext>

How to create a campaign?

<https://www.ofpec.com/editors-depot/index.php?action=details&id=341&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=25&game=OFP>

How to create a font?

<https://www.youtube.com/watch?v=JNQuyCt84RI>

How to make a texture?

Your image dimensions have to be a power of 2 (e.g. 256x256 or 128x64). Save it as TGA (RLE uncompressed) and then open in [TexView](#) and save it with paa or pac extension.

How to view textures?

Install [TexView2](#).

How to convert from or to WSS?

<https://www.ofpec.com/editors-depot/index.php?action=details&id=181&game=OFP>

How to create LIP file?

https://community.bistudio.com/wiki/Sound_Tools_Manual

https://ofp-faguss.com/files/official_editing_resources.zip

How to create OGG file?

https://community.bistudio.com/wiki/Multiplayer_Custom_Sounds_Tutorial#oggdropsXPd

How to create PBO file?

<http://ofpr.info.paradoxstudio.uk/utilities/299-make-pbo-1-1.html>

How to unpack PBO file?

<http://ofpr.info.paradoxstudio.uk/utilities/297-pbo-decryptor-1-5-by-amalfi.html>

<http://ofpr.info.paradoxstudio.uk/utilities/305-pbox-v1-0.html>

How to make models?

How to install Oxygen: <https://www.youtube.com/watch?v=Rce72rPMGu0>

Tutorials: https://ofp-faguss.com/files/official_editing_resources.zip

Tutorials: <https://ofp-faguss.com/addon/tutorials.zip>

How to make animations?

OFPAnim <https://lex-ofp.narod.ru/OFP/Tools/>

How to learn scripting?

https://www.youtube.com/watch?v=Veu3TNVTcEQ&list=PLdz-dynlxN37FT1JCh1I8PlmyBs6ENpu_

https://ofp-faguss.com/scripting_tutorials

How to make islands?

<https://www.wrptool.com/>

How to make missions?

https://community.bistudio.com/wiki/2D_Editor

<https://www.ofpec.com/editors-depot/index.php?action=details&id=1&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=33&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=30&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=23&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=22&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=61&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=325&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=326&game=OFP>

<https://www.ofpec.com/editors-depot/index.php?action=details&id=327&game=OFP>