Perfetto! Effettuerò una ricerca approfondita per creare una guida dettagliata e chiara su come scrivere script per GTA V utilizzando ScriptHookV .NET. La guida sarà in italiano e includerà spiegazioni passo-passo, esempi pratici e commenti per aiutare anche i principianti assoluti. Ti fornirò il contenuto direttamente in chat e, se possibile, lo convertirò in un PDF per una consultazione più comoda.

# Come creare script per GTA V con ScriptHookV .NET: Guida Introduttiva

Sviluppare mod e script per **Grand Theft Auto V** è possibile grazie a strumenti come *ScriptHookV* e *ScriptHookV .NET*. Questa guida dettagliata ti accompagnerà passo dopo passo nella creazione del tuo primo script in C# per GTA V, anche se **non hai alcuna esperienza di programmazione**. Imparerai come intercettare l'input da tastiera (es. premere un tasto per attivare un'azione), quali eventi usare, quali librerie installare e come strutturare il codice. Troverai inoltre esempi di codice completi e commentati (come mostrare messaggi a schermo, cambiare il modello del personaggio, far comparire un veicolo o teletrasportarsi) e i passaggi per testare e installare correttamente i tuoi script nel gioco. Infine, verranno fornite **risorse utili** per approfondire, tra cui link alla documentazione ufficiale, community e tutorial video.

**Nota:** Tutti i mod e script descritti qui funzionano **solo in modalità giocatore singolo (Story Mode)**. Le modifiche sono **vietate in GTA Online** e potrebbero portare al ban. Inoltre, a partire dal 2024 Rockstar Games ha introdotto il sistema anti-cheat *BattlEye*: assicurati di **disattivarlo dal Rockstar Launcher** (Impostazioni > BattlEye) prima di usare mod in single player (The Photographer's Guide to Los Santos).

## 1. Tasti ed eventi principali negli script di GTA V

Quando sviluppi uno script con ScriptHookV .NET, hai a disposizione alcuni **eventi chiave** forniti dalla classe base `GTA.Script`. I principali sono: **Tick**, **KeyDown**, **KeyUp** e **OnAbort**. Vediamo a cosa servono e come usarli:

- **Tick** – È un evento che viene chiamato ad ogni frame di gioco (in pratica, *in loop continuo* mentre lo script è attivo). Il codice che inserisci nel gestore di questo evento verrà eseguito ripetutamente, **ad ogni ciclo di gioco** (The Photographer's Guide to Los Santos). Questo è utile per logica continua, controlli costanti (es. monitorare la posizione del giocatore, applicare effetti costantemente, ecc.).

  ```
  private void OnTick(object sender, EventArgs e)
  {
      // Codice qui eseguito ad ogni frame di gioco
  }
  ```

- **KeyDown** – Viene chiamato quando il giocatore **preme** una tastiera specifica. In altre parole, ogni volta che un tasto viene abbassato, il tuo script può intercettarlo tramite questo evento (The Photographer's Guide to Los Santos) (The Photographer's Guide to Los Santos). All'interno del gestore, puoi verificare quale tasto è stato premuto e reagire di conseguenza. Ad esempio, potresti controllare se `F5` è stato premuto per attivare una funzione dello script.

```
private void OnKeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.F5)
    {
        // Codice eseguito quando il tasto F5 viene premuto
    }
}
```

*Nell'esempio sopra, `e.KeyCode` rappresenta il codice del tasto premuto, e viene confrontato con `Keys.F5` (definito in **System.Windows.Forms**). Se corrisponde, viene eseguito il blocco di codice desiderato.*

- **KeyUp** – Simile a KeyDown, ma viene chiamato quando il giocatore **rilascia** un tasto (The Photographer's Guide to Los Santos). Puoi usarlo per eseguire azioni al momento del rilascio invece che alla pressione. La struttura è analoga a KeyDown, ma tipicamente si usa `e.KeyCode` dentro un `OnKeyUp`. Questo evento è utile se vuoi rilevare combinazioni o fare qualcosa **dopo** che un tasto è stato premuto (ad esempio, conteggiare il tempo di pressione, ecc.).
- **OnAbort** – Viene invocato quando lo script viene terminato o scaricato (ad esempio durante un reload dei mod, o quando si chiude il gioco). Serve principalmente per fare **pulizia**: ad esempio, rimuovere oggetti o veicoli creati dallo script, resettare lo stato del gioco modificato, liberare risorse, ecc. (GitHub - VideoWarfare/LemonUICarSpawner: A simple car spawner using LemonUI.). Se il tuo script crea entità nel mondo di gioco (pedoni, veicoli, ecc.), è buona pratica distruggerle o rimuoverle in OnAbort, così da non lasciare "sporcizia" quando lo script si disattiva.

In uno **script base in C#**, questi eventi vengono normalmente gestiti iscrivendo dei metodi (handler) ad essi nel costruttore della classe script. Ad esempio, puoi iscrivere il metodo `OnTick` all'evento Tick, e così via, come mostrato nel codice seguente:

```
public class MioScript : Script
{
    public MioScript()
    {
        // Collegamento degli eventi principali agli handler
        Tick += OnTick;
        KeyDown += OnKeyDown;
        KeyUp += OnKeyUp;
        // (OnAbort non è un evento a cui ci si iscrive allo stesso modo,
        // ma si può eventualmente sovrascrivere un metodo OnAbort se
supportato)
    }

    private void OnTick(object sender, EventArgs e)
    {
        // ... codice da eseguire continuamente ...
    }

    private void OnKeyDown(object sender, KeyEventArgs e)
    {
        // ... codice da eseguire alla pressione di un tasto ...
    }

    private void OnKeyUp(object sender, KeyEventArgs e)
    {
        // ... codice da eseguire al rilascio di un tasto ...
```

```
    }
}
```

Come vedi, **ereditando dalla classe `Script`**, hai subito a disposizione Tick, KeyDown e KeyUp (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Basterà aggiungere le istruzioni desiderate nei rispettivi metodi. Nei prossimi esempi pratici (sezione 4) vedremo in dettaglio come utilizzare questi eventi per intercettare input (come premere F5 per attivare un'azione) e manipolare il gioco.

# 2. Librerie necessarie e differenze tra ScriptHookV e ScriptHookV .NET

Prima di iniziare a programmare, è importante capire **quali strumenti e librerie** servono e la differenza tra ScriptHookV "normale" e la versione .NET:

- **ScriptHookV (di Alexander Blade)** – È la libreria *base* che consente il modding di GTA V. Si tratta di un plugin nativo (in C++) che fa da **gancio (hook)** al motore di gioco, permettendo di eseguire chiamate "native" di GTA (cioè funzioni proprie del gioco) dall'esterno (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods). In pratica, ScriptHookV permette di creare mod in C++ compilate come *.asi* (simili a dei DLL speciali caricati dal gioco). Da solo, però, ScriptHookV funziona solo con mod scritte in C++.
- **ScriptHookV .NET (Community ScriptHookVDotNet)** – È un plugin aggiuntivo (anch'esso un file .asi) che si appoggia a ScriptHookV e integra un runtime .NET dentro GTA V (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods). Grazie a ScriptHookV .NET (abbreviato spesso in **SHVDN**), puoi scrivere script in linguaggi .NET come C# o VB.NET, che verranno eseguiti nel gioco. In sostanza, ScriptHookV .NET carica ed esegue tutti i file **.dll** (o anche file sorgenti .cs/.vb) presenti nella cartella `scripts` del gioco, permettendoti di scrivere mod usando il linguaggio C# anziché C++ (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods). ScriptHookV .NET mette a disposizione un'API ad oggetti (namespace **GTA**, **GTA.Native**, **GTA.Math**, ecc.) che semplifica l'interazione col gioco.
- **Differenze principali:** ScriptHookV è indispensabile per qualsiasi modding su GTA V (è il gancio di base per le funzioni native), mentre ScriptHookV .NET è una *estensione* per chi preferisce usare C# e altri linguaggi .NET. Se vuoi programmare in C#, userai **entrambi**: ScriptHookV *deve* essere installato, e poi aggiungerai ScriptHookV .NET per caricare i tuoi script .NET ( Community Script Hook V .NET - GTA5-Mods.com ) ( Community Script Hook V .NET - GTA5-Mods.com ). Un'altra alternativa esistente è **RAGE Plugin Hook (RPH)**, un altro hook sviluppato separatamente, usato soprattutto da mod avanzate come LSPD:FR. RAGE Plugin Hook consente anch'esso di creare plugin in .NET (e supporta anche C++/CLI), ma è meno usato per mod generiche perché richiede di avviare il gioco attraverso il suo launcher e offre funzionalità più specifiche (ad es. gestione avanzata di chiamate di gioco, utili per mod molto complesse) (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods). In generale, la maggior parte delle mod single-player utilizza la combinazione ScriptHookV + ScriptHookV .NET, perché è più semplice da installare e usare.

- **Librerie e dipendenze richieste:** Per far girare ScriptHookV .NET, assicurati di avere installato **.NET Framework 4.8** (su Windows 10 è già presente) e il **Microsoft Visual C++ Redistributable 2019 x64** ( Community Script Hook V .NET - GTA5-Mods.com ). Queste componenti sono necessarie affinché gli script .NET funzionino correttamente. Inoltre, quando svilupperai il tuo script in C#, dovrai fare riferimento a diverse librerie:
    - `ScriptHookVDotNet3.dll` (o la versione stabile più recente) – contiene le classi GTA, Script, ecc. È la libreria principale dell'API .NET di GTA V.
    - **System.Windows.Forms** – libreria .NET standard che fornisce, tra le altre cose, l'enumerazione `Keys` per riconoscere i tasti (es. Keys.F5, Keys.H, ecc.). Servirà per gestire gli input da tastiera nei tuoi script.
    - **System.Drawing** – libreria .NET standard per colori, rettangoli, ecc. Viene spesso usata per specificare colori (ad esempio colore dei veicoli, testo, ecc.).
    - **GTA.Native** – namespace (spazio dei nomi) incluso in ScriptHookVDotNet per accedere alle funzioni native di GTA V e a enumerazioni come `PedHash` (elenco dei modelli pedoni), `VehicleHash` (modelli veicoli), `WeaponHash` (armi), ecc. Userai questo namespace quando vorrai chiamare direttamente funzioni del gioco non coperte dalle classi ad alto livello.
    - **GTA.Math** – namespace che contiene strutture matematiche utili, come `Vector3` (coordinate 3D) e altre, per gestire posizioni, rotazioni e vettori nel mondo di gioco.

# Installazione di ScriptHookV e ScriptHookV .NET

Per poter eseguire i tuoi script, devi assicurarti di aver installato correttamente sia ScriptHookV che ScriptHookV .NET:

1. **Installa ScriptHookV (base):** Scarica l'ultima versione dal sito ufficiale di Alexander Blade (dev-c.com) o da una fonte affidabile. Nella zip troverai dei file, tra cui *ScriptHookV.dll* e *dinput8.dll*. Copia **ScriptHookV.dll** e **dinput8.dll** nella cartella principale di GTA V (dove c'è GTAV.exe) (The Photographer's Guide to Los Santos). *Nota:* dinput8.dll è l'ASI Loader che permette di caricare i plugin .asi come ScriptHookV stesso e altri mod.

2. **Installa Community ScriptHookV .NET:** Scarica l'ultima versione (stabile o nightly) da GitHub o da GTA5-Mods.com ( Community Script Hook V .NET - GTA5-Mods.com ). Estrai i file e copia nella cartella principale di GTA V i seguenti: **ScriptHookVDotNet.asi**, **ScriptHookVDotNet2.dll** e **ScriptHookVDotNet3.dll** ( Community Script Hook V .NET - GTA5-Mods.com ) ( Community Script Hook V .NET - GTA5-Mods.com ). Assicurati di aver creato una cartella **`scripts`** (tutto minuscolo) nella directory di GTA V (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) – è lì che andranno i tuoi script compilati (.dll) o eventuali script in sorgente (.cs/.vb). Se la cartella non c'è, creala manualmente.

3. **(Facoltativo) Versioni Nightly:** Tieni presente che le versioni stabili di ScriptHookVDotNet a volte potrebbero non essere compatibili con l'ultimissima patch di GTA V. In quel caso, gli sviluppatori forniscono delle *nightly builds* (versioni in sviluppo) compatibili. Ad esempio, finché la versione 3.7.0 stabile non viene rilasciata, è consigliato usare la nightly v3.6.0 per funzionare con le patch più recenti (Getting Started · scripthookvdotnet/scripthookvdotnet

Wiki · GitHub). Puoi trovare le nightly nella sezione release su GitHub. Per un principiante, va bene iniziare con l'ultima stabile; se però gli script .NET non dovessero caricare, potrebbe essere necessario passare a una nightly più aggiornata.

Una volta installato il tutto, avvia GTA V in modalità storia. Se hai installato correttamente ScriptHookV e .NET, premendo **F4** durante il gioco si dovrebbe aprire la *console di ScriptHookV .NET* (The Photographer's Guide to Los Santos). Puoi digitare ad esempio il comando `Help()` nella console e premere Invio per vedere la lista di comandi disponibili, oppure `Reload()` per ricaricare gli script (ne parleremo più avanti). Se la console appare, significa che l'ambiente per gli script .NET è pronto.

# 3. Struttura base di uno script C# per GTA V

Vediamo ora come **creare da zero un progetto C#** per uno script di GTA V e la struttura fondamentale del codice. Useremo Visual Studio (va bene la versione Community, gratuita) come ambiente di sviluppo.

(The Photographer's Guide to Los Santos) *Figura: Creazione di un nuovo progetto in Visual Studio. Seleziona **Class Library (.NET Framework)** per creare una libreria .dll compatibile con ScriptHookV .NET.* (The Photographer's Guide to Los Santos)

**Passo 1 – Creare il progetto C#:** Apri Visual Studio e crea un nuovo progetto selezionando **"Class Library (.NET Framework)"** come modello (non usare .NET Core o .NET 5+, poiché ScriptHookV .NET supporta solo .NET Framework) (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Assegna un nome al progetto (es. *GTA5ScriptTest*) e conferma. Visual Studio genererà un progetto vuoto con una classe di esempio (es. Class1.cs).

**Passo 2 – Aggiungere riferimenti alle librerie necessarie:** Una volta creato il progetto, dobbiamo riferire le librerie di ScriptHookV .NET e le dipendenze .NET standard:

- In Visual Studio, vai nel *Solution Explorer* (lato destro), fai clic col destro su **References** (Riferimenti) del progetto e scegli *Add Reference…* (The Photographer's Guide to Los Santos).
- Nella finestra di gestione riferimenti, clicca su **Browse** e seleziona il file **ScriptHookVDotNet3.dll** (lo trovi nella cartella di GTA V dove l'hai copiato precedentemente) (The Photographer's Guide to Los Santos). Aggiungi anche *ScriptHookVDotNet2.dll* se presente, per compatibilità (non strettamente necessario se usi solo l'API v3, ma potrebbe essere utile avere entrambi come riferimento).
- Sempre nella sezione Assemblies, cerca e aggiungi **System.Windows.Forms** e **System.Drawing** (The Photographer's Guide to Los Santos). (Forms ci serve per l'enum Keys e gestione input, Drawing per eventuali colori/grafica).
- Conferma l'aggiunta dei riferimenti. Ora nel tuo progetto dovresti vedere sotto "References" le librerie aggiunte.

**Passo 3 – Configurare il progetto per facile deploy (facoltativo):** Durante lo sviluppo, è comodo che la .dll compilata venga copiata automaticamente nella cartella `scripts` di GTA V. Puoi impostare un evento post-build che faccia ciò (The Photographer's Guide to Los Santos). Ad

esempio, in Visual Studio vai su *Project > Properties > Build Events*, e in *Post-build event command line* inserisci un comando `xcopy`:

```
xcopy /Y "$(TargetDir)$(ProjectName).dll" "C:\Percorso\Alla\Cartella\GTA V\
scripts\"
```

Sostituisci il percorso con quello reale della tua directory di gioco. Questo comando copierà la DLL compilata (dalla cartella bin\Debug) direttamente nella cartella scripts ad ogni build, evitandoti di farlo a mano ogni volta (The Photographer's Guide to Los Santos). (Ricorda di creare la cartella scripts se non l'hai già fatta).

**Passo 4 – Scrivere la classe dello script:** Apri il file Class1.cs (o come l'hai nominato) e inizia a scrivere il codice del tuo script. La struttura base è:

- *Namespace:* opzionale, puoi definirne uno personalizzato o usare quello di default.
- *Using:* includi gli spazi dei nomi necessari:

  ```
  using System;
  using System.Windows.Forms;
  using System.Drawing;
  using GTA;
  using GTA.Math;
  using GTA.Native;
  ```

  Questi coprono le funzioni base .NET e l'API di ScriptHookV .NET.
- *Definizione classe:* crea una classe pubblica che **eredita da `Script`** (GTA.Script). Ad esempio:

  ```
  public class MyFirstScript : Script
  {
      // ...
  }
  ```

- *Costruttore:* all'interno della classe, definisci un costruttore (metodo con nome uguale alla classe) pubblico. In questo costruttore collegherai gli eventi Tick, KeyDown, KeyUp ai tuoi gestori (metodi) come visto nella sezione 1. Esempio:

  ```
  public MyFirstScript()
  {
      Tick += OnTick;
      KeyDown += OnKeyDown;
      KeyUp += OnKeyUp;
      // (eventualmente altre inizializzazioni)
  }
  ```

- *Metodi evento:* sempre dentro la classe, definisci i metodi `OnTick`, `OnKeyDown` e `OnKeyUp` con le firme esatte (`object sender, EventArgs e` per Tick e `object sender, KeyEventArgs e` per gli eventi tastiera). Inizialmente, puoi lasciarli vuoti o metterci qualche test (come un messaggio di log). Questi metodi verranno chiamati automaticamente quando il gioco gira (Tick) o quando premi/rilasci tasti.

Per esempio, ecco come appare uno *scheletro* di script completo:

```
using System;
using System.Windows.Forms;
using GTA;
using GTA.Native;
```

```
using GTA.Math;

public class MyFirstScript : Script
{
    public MyFirstScript()
    {
        // Collega gli eventi del gioco agli handler
        Tick += OnTick;
        KeyDown += OnKeyDown;
        KeyUp += OnKeyUp;
    }

    private void OnTick(object sender, EventArgs e)
    {
        // Codice eseguito ogni frame
    }

    private void OnKeyDown(object sender, KeyEventArgs e)
    {
        // Codice eseguito alla pressione di un tasto
    }

    private void OnKeyUp(object sender, KeyEventArgs e)
    {
        // Codice eseguito al rilascio di un tasto
    }
}
```

Questa sarà la base su cui costruiremo le funzionalità desiderate. Nei prossimi esempi popoleremo questi metodi per fare cose concrete (mostrare messaggi, cambiare modello, ecc.).

**Passo 5 – Compilazione e test rapido:** Premi **F6 (Build)** in Visual Studio per compilare il progetto. Se hai impostato il post-build, la DLL risultante sarà automaticamente copiata nella cartella `scripts` di GTA V (The Photographer's Guide to Los Santos). Altrimenti, prendi manualmente il file .dll generato (lo trovi in `bin\Debug\Tuoprogetto.dll`) e copialo nella cartella `scripts`. Ora avvia GTA V (modalità storia), e verifica il funzionamento dello script: se hai, ad esempio, inserito un messaggio nel Tick o un’azione su un tasto, prova a vedere se accade. Più avanti discuteremo come usare la console per ricaricare gli script al volo durante i test.

# 4. Esempi pratici di script (codice completo e commentato)

In questa sezione realizzeremo **quattro esempi di script completi**, ciascuno con un obiettivo specifico, per mettere in pratica quanto appreso:

1. **Script base che mostra un messaggio a schermo.**
2. **Script che cambia il modello del personaggio (skin).**
3. **Script che spawna (crea) un veicolo davanti al giocatore.**
4. **Script che teletrasporta il giocatore a una certa posizione.**

Ogni esempio sarà una classe separata con il proprio costruttore e metodi. Puoi inserirli anche tutti in un unico progetto, in file .cs diversi (ognuno sarà compilato come script separato) oppure provarli uno alla volta. Il codice è ampiamente commentato per spiegare cosa fa ogni riga.

> **Nota:** Prima di provare questi script, assicurati di aver seguito i passi di installazione e di avere la struttura base funzionante. Ricorda di aggiungere le direttive `using`

necessarie (System, GTA, etc.) come mostrato prima, altrimenti i riferimenti alle classi (come `GTA.UI` o `World`) potrebbero non essere risolti.

## 4.1 Script base: mostrare un messaggio sullo schermo

Iniziamo con qualcosa di molto semplice: un mod che, appena caricato, mostra un messaggio sullo schermo per confermare che lo script sta funzionando. È l'equivalente del classico "Hello, world".

**Cosa fa:** Appena avviato (o ricaricato), lo script visualizza una notifica in alto a sinistra dello schermo con un testo di saluto.

**Implementazione:** Possiamo ottenere ciò sfruttando la classe `GTA.UI.Notification` fornita dall'API .NET di GTA V. Essa permette di creare le stesse notifiche stile GTA (quelle che appaiono sopra la minimappa). Non avremo bisogno di Tick o input per questo esempio, basta eseguire il codice una volta all'avvio dello script (quindi direttamente nel costruttore).

Ecco il codice completo:

```
using System;
using GTA;
using GTA.UI;

public class HelloScript : Script
{
    public HelloScript()
    {
        // Mostra una notifica di testo sullo schermo per 5 secondi
        Notification.Show("~y~Ciao~w~! Script attivato correttamente.");
        // ~y~ e ~w~ sono tag di colore (giallo e bianco) usati da GTA per il
testo
    }
}
```

**Spiegazione delle linee:**

- `using GTA.UI;` – Include il namespace per le funzionalità UI (User Interface) di ScriptHookV .NET, che contiene la classe `Notification`.
- La classe `HelloScript` estende `Script`. Nel costruttore, chiamiamo `Notification.Show(...)` (Class Notification | Community Script Hook V .NET ) per visualizzare una notifica. In questo caso stiamo passando una stringa, che può contenere anche codici di formattazione di GTA (i tag `~ ~` per colori: `~y~` rende il testo giallo, `~w~` bianco, ecc.). La notifica comparirà per qualche secondo come messaggio di gioco.
- Non usiamo eventi Tick o input qui, poiché vogliamo solo un messaggio all'avvio. ScriptHookV .NET esegue il costruttore una volta quando carica lo script, quindi questa notifica apparirà quando entri in gioco o ricarichi gli script. È un semplice test per verificare che tutto sia configurato bene.

Compila e copia questo script (DLL) nella cartella *scripts*, quindi avvia GTA V. Dovresti vedere comparire il messaggio di benvenuto in alto a sinistra poco dopo il caricamento del mondo di gioco.

## 4.2 Script: cambiare il modello del personaggio (skin change)

In questo esempio, vedremo come trasformare il protagonista in un altro modello di pedone/animale. Ad esempio, premendo un tasto, il nostro personaggio diventerà un **gatto** (sì, GTA

V include anche modelli di animali!). Questo illustra l'uso delle funzioni per cambiare *skin* del player.

**Cosa fa:** Quando il giocatore preme il tasto **H** sulla tastiera, lo script cambia il modello del giocatore corrente in quello di un gatto. In altre parole, trasforma il protagonista in un gatto nel mondo di gioco. (Puoi scegliere altri modelli: useremo il gatto per dimostrazione.)

**Dettagli tecnici:** GTA V ha tanti modelli di ped disponibili, identificati da hash o enum. ScriptHookV .NET fornisce l'enumerazione `PedHash` (nel namespace GTA.Native) con i nomi di molti modelli. Tra questi c'è `PedHash.Cat` per il gatto. Cambiare il modello del giocatore richiede due passi:

1. **Caricare il modello in memoria.** Bisogna assicurarsi che il modello 3D del gatto sia caricato dal gioco (altrimenti il cambio non avrà effetto o potrebbe causare ritardi).
2. **Applicare il modello al giocatore.** Si usa il metodo `Game.Player.ChangeModel(...)` passando il modello desiderato (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo metodo crea un nuovo ped per il player con quel modello.

Implementeremo il tutto nel metodo OnKeyDown, reagendo alla pressione del tasto H.

Ecco il codice completo e commentato:

```
using System;
using System.Windows.Forms;
using GTA;
using GTA.Native;

public class ChangeSkinScript : Script
{
    public ChangeSkinScript()
    {
        KeyDown += OnKeyDown; // Ascolta la pressione dei tasti
    }

    private void OnKeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.H) // Se il tasto premuto è 'H'
        {
            // 1. Creiamo un oggetto Model per il ped "Cat" (gatto)
            Model catModel = new Model(PedHash.Cat);
            // 2. Richiediamo il caricamento del modello in memoria (timeout 500
ms)
            catModel.Request(500);

            // 3. Verifichiamo che il modello esista nel gioco ed è valido
            if (catModel.IsInCdImage && catModel.IsValid)
            {
                // 4. Aspettiamo finché il modello non è completamente caricato
                while (!catModel.IsLoaded) Script.Wait(50);
                // 5. Cambiamo il modello del giocatore attuale al modello del
gatto
                Game.Player.ChangeModel(catModel);
                // (Opzionale) Reimposta i vestiti predefiniti se il modello ha
componenti abbigliamento
                Game.Player.Character.Style.SetDefaultClothes();
            }
```

```
            // 6. Segnala al gioco che il modello non serve più in memoria
(verrà scaricato quando opportuno)
            catModel.MarkAsNoLongerNeeded();
        }
    }
}
```

**Spiegazione passo-passo:**

- Nel costruttore colleghiamo solo l'evento `KeyDown` al metodo `OnKeyDown` (non serve Tick qui).
- Quando un tasto viene premuto, controlliamo se è `Keys.H`. Se **H** è premuto, iniziamo la procedura di cambio modello.
- Creiamo un oggetto `Model` passando `PedHash.Cat`. Questo prepara un riferimento al modello del gatto. (PedHash è un enum di GTA.Native con vari ped, includendo animali).
- Chiamiamo `Request(500)` sul modello: questo dice al gioco di caricare quel modello entro 500 ms (mezzo secondo) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo evita di tentare di usare un modello non ancora disponibile.
- Controlliamo `catModel.IsInCdImage && catModel.IsValid`: queste proprietà verificano che il modello esista nei file di gioco. `IsInCdImage` indica se il modello è nel gioco (non un modello custom mancante), `IsValid` controlla se l'oggetto Model è valido. Se entrambe vere, procediamo.
- Entriamo in un piccolo loop `while (!catModel.IsLoaded) Script.Wait(50);` – questo fa attendere il nostro script finché il modello non risulta caricato in memoria. `Script.Wait(50)` sospende lo script per 50 ms per evitare blocchi (il gioco nel frattempo carica il modello in background).
- Una volta caricato, chiamiamo `Game.Player.ChangeModel(catModel);` (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo metodo crea un nuovo ped per il giocatore con il modello specificato (in questo caso il gatto). In pratica, *sostituisce* il personaggio attuale con un gatto. Subito dopo, per sicurezza, resettiamo i vestiti di default con `Game.Player.Character.Style.SetDefaultClothes();` (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo serve nei casi di modelli umani per applicare abiti predefiniti; per un gatto probabilmente non cambia nulla, ma lo lasciamo come buona pratica.
- Infine, `catModel.MarkAsNoLongerNeeded();` consente al gioco di liberare quel modello dalla memoria quando lo riterrà opportuno, dal momento che l'abbiamo utilizzato. Non lo rimuove immediatamente (il giocatore è ancora un gatto!) ma indica che non ci serve tenerlo in cache perché, se il giocatore cambiasse zona, il modello potrà essere scaricato se non in uso altrove.

Ora, compilando e caricando questo script, quando premi **H** durante il gioco, dovresti vedere il tuo personaggio trasformarsi in un gatto (The Photographer's Guide to Los Santos) (The Photographer's Guide to Los Santos). Per tornare normale, puoi o usare un altro script per cambiare di nuovo il modello (es. Game.Player.ChangeModel(PedHash.Franklin) per tornare Franklin, se eri Franklin), oppure semplicemente ricaricare gli script (premi il tasto reload o usa la console `Reload()` – il gioco ripristinerà il modello originale se stavi usando un personaggio della storia, altrimenti potrebbe lasciarti col gatto finché non cambi sessione).

### 4.3 Script: spawnare un veicolo davanti al giocatore

Questo esempio mostra come creare (spawnare) un veicolo nel mondo di gioco via script, di fronte al personaggio. È una funzionalità tipica di molti trainer/mod menu (es: “spawn car”). Lo legheremo alla pressione di un tasto (ad es. **F5**) per attivarlo.

**Cosa fa:** Quando il giocatore preme **F5**, un veicolo (in particolare una supercar *Zentorno*) verrà generato a qualche metro di distanza di fronte al protagonista. Inoltre, lo script applicherà al veicolo alcune personalizzazioni: gomme antiproiettile, colore personalizzato e targa personalizzata.

**Come funziona:** Utilizzeremo il metodo `World.CreateVehicle(...)` fornito dall’API (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo metodo consente di creare un veicolo dato un modello (identificato da `VehicleHash` o da un hash numerico) e una posizione. Per il modello useremo l’enum `VehicleHash.Zentorno` (una supercar del gioco), per la posizione prenderemo quella del giocatore più un offset in avanti. Una volta creato l’oggetto `Vehicle`, potremo impostarne proprietà (come colore, targa, ecc.) tramite i suoi membri.

Ecco il codice completo:

```
using System;
using System.Drawing;
using System.Windows.Forms;
using GTA;
using GTA.Math;
using GTA.Native;

public class SpawnCarScript : Script
{
    public SpawnCarScript()
    {
        KeyUp += OnKeyUp; // Usiamo KeyUp in questo caso, ma va bene anche
KeyDown
    }

    private void OnKeyUp(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.F5) // attiva con tasto F5
        {
            Ped player = Game.Player.Character;
            // Calcola la posizione di spawn: 3 metri davanti al giocatore
            Vector3 spawnPos = player.Position + player.ForwardVector * 3.0f;
            // Calcola l'angolo (heading) per far rivolgere l'auto verso destra
del player
            float heading = player.Heading + 90.0f;

            // Crea il veicolo Zentorno
            Vehicle vehicle = World.CreateVehicle(VehicleHash.Zentorno,
spawnPos, heading);
            // Controllo: se la creazione è fallita (vehicle è null), esci
            if (vehicle == null) return;

            // Impostazioni sul veicolo creato:
            vehicle.CanTiresBurst = false; // Gomme antiproiettile (non
scoppiano)
            vehicle.Mods.CustomPrimaryColor = Color.FromArgb(38, 38, 38); //
Colore primario grigio scuro
            vehicle.Mods.CustomSecondaryColor = Color.Orange; // Colore
secondario arancione
            vehicle.PlaceOnGround(); // Assicura che l'auto sia posata sul
terreno
```

```
            vehicle.NumberPlate = "SHVDN"; // Targa personalizzata
        }
    }
}
```

**Spiegazione delle righe più importanti:**

- Nel costruttore colleghiamo l'evento `KeyUp` al nostro metodo OnKeyUp. (Usiamo KeyUp perché magari vogliamo che l'azione avvenga al rilascio del tasto, per evitare possibili ripetizioni multiple se il frame rate è basso e tieni premuto; comunque KeyDown avrebbe funzionato simile in questo caso).
- Quando il giocatore rilascia un tasto, controlliamo se è F5. Se sì, procediamo con lo spawn.
- `Ped player = Game.Player.Character;` ottiene l'oggetto Ped corrispondente al personaggio controllato dal giocatore.
- `spawnPos = player.Position + player.ForwardVector * 3.0f;` – calcola una posizione 3 unità (metri) **davanti** al giocatore. `player.Position` è la posizione attuale del player (un Vector3 con coordinate x,y,z). `player.ForwardVector` è un vettore direzionale che punta davanti al ped. Moltiplicandolo per 3 otteniamo un vettore che va 3 metri in avanti. Sommato alla posizione attuale dà la coordinata finale di spawn, così l'auto non compare esattamente addosso al player ma un po' più avanti.
- `float heading = player.Heading + 90.0f;` – calcoliamo l'orientamento (angolo in gradi) che daremo all'auto. `player.Heading` è la direzione in cui il giocatore sta guardando (0-360 gradi). Aggiungendo 90 gradi, stiamo ruotando di lato (ad esempio, se il player guarda nord, l'auto sarà rivolta ad est). In questo caso vogliamo solo differenziare l'orientamento per estetica, facciamo finta di parcheggiare l'auto di fianco a noi.
- `Vehicle vehicle = World.CreateVehicle(VehicleHash.Zentorno, spawnPos, heading);` – questa chiamata crea il veicolo (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Passiamo il modello (Zentorno), la posizione e l'heading. Se la creazione riesce, restituisce un oggetto `Vehicle` che rappresenta il veicolo nel gioco.
- Subito dopo controlliamo `if (vehicle == null) return;`. In casi rari CreateVehicle può fallire (ad esempio modello non valido). In tal caso meglio uscire dalla funzione per evitare errori successive su un oggetto nullo.
- Ora configuriamo qualche proprietà del veicolo:
    - `vehicle.CanTiresBurst = false;` – Impedisce che le gomme possano scoppiare (tipicamente come le ruote antiproiettile).
    - `vehicle.Mods.CustomPrimaryColor = Color.FromArgb(38, 38, 38);` – Imposta un colore primario personalizzato. Usiamo un grigio scuro specificando il colore con componente RGBA (qui 38,38,38 con alpha default 255). `Mods` dà accesso alle modifiche e tuning del veicolo.
    - `vehicle.Mods.CustomSecondaryColor = Color.Orange;` – Colore secondario arancione (usiamo un colore predefinito dall'enum Color di .NET).
    - `vehicle.PlaceOnGround();` – Questa funzione sposta leggermente il veicolo verso il basso finché non tocca terra. Serve perché se la posizione di spawn era leggermente sopra il suolo, l'auto potrebbe rimanere sospesa; con questa chiamata la

"appoggiamo" al suolo (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub).

- `vehicle.NumberPlate = "SHVDN";` – Imposta la targa del veicolo alla stringa desiderata (massimo 8 caratteri). In questo caso "SHVDN" (acronimo di ScriptHookV Dot Net).

Ora compila e testa questo script. Avviando il gioco e premendo **F5**, dovresti vedere comparire una Zentorno vicino a te, con i colori impostati e la targa personalizzata (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Puoi salire a bordo e guidarla subito. Ogni volta che premi F5 ne spawnerai un'altra (attenzione a non esagerare per non riempire l'area di veicoli; eventualmente potresti migliorare lo script per cancellare la precedente, ma teniamolo semplice per ora).

## 4.4 Script: teletrasportare il giocatore

Ultimo esempio: come spostare il giocatore in un punto specifico della mappa, ovvero effettuare un **teletrasporto**. Questa funzione è utile per creare mod di viaggio rapido, o per testare posizioni. Nel nostro caso, legheremo il teletrasporto al tasto **F7**.

**Cosa fa:** Premendo F7, lo script teletrasporta immediatamente il personaggio giocante a una posizione predefinita (in questo esempio una località nel deserto di Grand Senora). Inoltre, mostra un messaggio di conferma al giocatore.

**Come funziona:** Teletrasportare il ped del giocatore è molto semplice usando ScriptHookV .NET: basta modificare la proprietà `Position` del ped o usare il metodo specifico `SetPosition(...)`. Impostando una nuova `Position` (un Vector3 con coordinate X, Y, Z), il gioco sposterà quell'entità istantaneamente a quelle coordinate (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Bisogna solo assicurarsi che le coordinate siano valide (es: sopra il terreno, non sottoterra). Nel nostro esempio, useremo coordinate note e ragionevoli. (In applicazioni avanzate, si potrebbe teletrasportare al segnalino sulla mappa, ma ciò richiederebbe calcolare la Z del terreno con una funzione nativa `GET_GROUND_Z_FOR_3D_COORD` – qui per semplicità usiamo coordinate fisse.)

Ecco il codice:

```
using System;
using System.Windows.Forms;
using GTA;
using GTA.Math;
using GTA.UI;

public class TeleportScript : Script
{
    public TeleportScript()
    {
        KeyDown += OnKeyDown;
    }

    private void OnKeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.F7) // tasto F7 attiva il teletrasporto
        {
            // Coordinate bersaglio (esempio: nel deserto vicino a Sandy Shores)
            Vector3 targetPos = new Vector3(402.807f, 3175.139f, 53.14062f);
```

```
            // Imposta la posizione del giocatore a queste coordinate
            Game.Player.Character.Position = targetPos;
            // Notifica di conferma a schermo
            Notification.Show($"Teletrasporto eseguito a X:{targetPos.X:F1} Y:
{targetPos.Y:F1} Z:{targetPos.Z:F1}");
        }
    }
}
```

**Spiegazione:**

- Nel costruttore colleghiamo l'evento KeyDown all'handler.
- Alla pressione di un tasto, controlliamo se è F7. Se sì, definiamo un `Vector3 targetPos` con le coordinate di destinazione. In questo caso usiamo (402.807, 3175.139, 53.14) che corrispondono a una zona pianeggiante nel deserto di Grand Senora (nei pressi dell'aeroporto di Trevor, Sandy Shores) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub).
- `Game.Player.Character.Position = targetPos;` – questa è la linea chiave: assegnando una nuova posizione al ped del giocatore, lo teletrasportiamo immediatamente lì (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Il gioco aggiornerà la posizione e, se quell'area non era caricata, potrebbe caricarla al volo (potresti vedere un leggero freeze se è molto lontano).
- Infine, usiamo `Notification.Show()` (come nel primo esempio) per mostrare un messaggio di conferma al giocatore, includendo le coordinate. Qui ho usato una stringa interpolata con formattazione `{targetPos.X:F1}` per mostrare X con 1 decimale, giusto per esempio.

Compila e prova il teletrasporto. Vai in una zona qualsiasi della mappa nel gioco, premi **F7**, e dovresti ritrovarti nel deserto (vedrai attorno a te il paesaggio cambiato istantaneamente). La notifica confermerà le coordinate arrivo. Se il teletrasporto ti mette in un luogo alto nel cielo o sottoterra: in questo caso abbiamo scelto intenzionalmente coordinate sul terreno, ma se sbagli coordinate potresti apparire in aria (il personaggio cadrà a terra) o dentro oggetti/edifici. Fare sempre attenzione alle Z. In mod avanzati, come detto, spesso si calcola la quota del suolo dinamicamente o si aggiunge un po' di offset verticale e poi si lascia cadere il player.

# 5. Test, installazione e troubleshooting degli script

Una volta scritto il codice e compilato il tuo script (.dll), come lo provi nel gioco? In questa sezione ricapitoliamo i passaggi per **installare** e testare gli script in GTA V, e forniamo alcuni consigli per il debugging e la risoluzione dei problemi comuni.

**Posizionamento della DLL:** Dopo aver compilato il tuo progetto in Visual Studio (Build successo), assicurati che il file `.dll` risultante sia nella cartella `scripts` di GTA V. Ad esempio, se hai creato *MyScript.dll*, questo deve trovarsi in `Grand Theft Auto V\scripts\MyScript.dll`. Se hai seguito la sezione 3 e impostato l'evento post-build, la copia dovrebbe essere avvenuta automaticamente (The Photographer's Guide to Los Santos). Altrimenti, copia manualmente il file ogni volta che ricompili nuove modifiche.

**Avvio del gioco:** Avvia GTA V in modalità Storia. Durante il caricamento, ScriptHookV .NET caricherà automaticamente tutti i .dll presenti nella cartella *scripts*. Se uno script ha un costruttore

con operazioni immediate (come il nostro HelloScript che mostra un messaggio), potresti vedere l'effetto già appena entri in gioco (il messaggio di benvenuto). Per altri script in attesa di input, dovrai attivare i relativi tasti in gioco.

**Testare le funzionalità:** Prova a premere i tasti che hai programmato (F5, F7, H, etc.) e verifica che accada quanto previsto (spawn dei veicoli, teletrasporto, ecc.). Se qualcosa non funziona:

- Assicurati di non avere conflitti di tasti: ad esempio, per default **F4** apre la console di ScriptHookV .NET, quindi evita di usare F4 per le tue azioni. I tasti F5–F7 di solito sono liberi. Anche evitare tasti che il gioco usa per funzioni importanti (ad es. E per interagire, se usato in missioni, ecc.). Nel dubbio, preferisci tasti funzione o combinazioni poco usate.
- Se premi il tasto ma non succede nulla, potrebbe essere che lo script non è stato caricato affatto (vedi se altri script funzionano per capire se SHVDN è attivo). In tal caso, vedi la sezione troubleshooting sotto.

**Ricaricare gli script senza riavviare il gioco:** Durante lo sviluppo, è scomodo dover riavviare GTA V ogni volta. Fortunatamente ScriptHookV .NET permette di **ricaricare** gli script al volo. Puoi farlo in due modi:

- Apri la console di ScriptHookV .NET premendo **F4**, quindi digita il comando `Reload()` e premi Invio (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo scaricherà e ricaricherà tutte le DLL nella cartella scripts (applicando eventuali modifiche nel codice che hai ricompilato nel frattempo).
- In alternativa, è presente un tasto di reload rapido (di default spesso è il tasto **Insert** sulla tastiera). Premendo *Ins* in gioco dovresti vedere un messaggio tipo "scripts reloaded". (Se non funziona, usa la console come sopra o controlla nel file *ScriptHookVDotNet.ini* quale tasto è assegnato a "ReloadKey").

La capacità di ricaricare sul momento è molto utile: puoi ad esempio modificare il tuo codice (e.g. cambiare un numero, aggiungere un print) in Visual Studio **mentre il gioco è aperto in background**, compilare (assicurati che la DLL aggiornata venga copiata), tornare in GTA V, fare Reload(), e provare subito la modifica (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo velocizza tantissimo il ciclo di sviluppo.

**Debug e messaggi di log:** Per capire cosa fa il tuo script o diagnosticare problemi, hai vari strumenti:

- Puoi usare la console in-game (tasto F4) per eseguire comandi e leggere eventuali messaggi di errore. Ad esempio, se il tuo script lancia un'eccezione (un errore non gestito), ScriptHookV .NET di solito stamperà un messaggio di errore in console con il nome dell'eccezione e la linea incriminata.
- Usa i metodi di output su console: c'è la funzione `GTA.UI.Notify` per messaggi a schermo, ma anche la possibilità di stampare sulla console o su file log. Un modo rapido è usare `Console.WriteLine("messaggio")` nel codice C#: ScriptHookV .NET redireziona l'output della console standard sul suo log (e se la console in-game è aperta, lo vedi lì). Quindi puoi inserire dei Console.WriteLine per tracciare valori o segnare se un pezzo di codice è stato raggiunto.

- Controlla il file di log: nella cartella principale di GTA V, ScriptHookVDotNet crea dei file di log, tipicamente chiamati `ScriptHookVDotNet.log` (o con un numero, es. ScriptHookVDotNet2.log, a seconda della versione). Apri questo file con un editor di testo dopo aver eseguito il gioco: contiene l'elenco degli script caricati e eventuali errori (stacktrace) occorsi. È molto utile per capire perché un certo script non parte (ad esempio, potrebbe mancare una reference, o c'è un errore sintattico che non hai notato).
- **Errori comuni:** Alcuni problemi che potresti incontrare:
    - Lo script non viene caricato affatto: verifica di aver compilato come .NET Framework (non .NET Core) e come DLL, e che la .dll sia nella cartella `scripts`. Se la dll è mancante o in altra cartella, SHVDN non la trova.
    - Conflitti di versioni: se hai usato una funzione disponibile solo in SHVDN v3 ma hai caricato solo ScriptHookVDotNet2.dll, potrebbe non funzionare. Assicurati di usare ScriptHookVDotNet3.dll come riferimento e di avere l'ASI e i DLL corrispondenti in gioco.
    - **Access Denied o mancati caricamenti:** Alcuni utenti hanno segnalato di dover avviare GTA V come amministratore affinché i log vengano creati o gli script caricati. In generale non dovrebbe servire, ma se non vedi proprio file log, prova questa strada.
    - Crash del gioco: se GTA V crasha all'avvio dopo aver messo un nuovo script, probabilmente c'è qualcosa di gravemente sbagliato (es. hai referenziato una libreria esterna incompatibile, o usi un mod che confligge). Rimuovi lo script aggiunto e verifica se il crash sparisce. Controlla di avere le versioni aggiornate di ScriptHookV (dopo ogni aggiornamento del gioco, spesso esce un nuovo ScriptHookV – usare uno vecchio causa crash immediati con errore "critical error") e di ScriptHookVDotNet compatibile.

**Distribuzione del tuo mod:** Se decidi di condividere il tuo script con altri, generalmente dovrai fornire il file .dll (compilato in *Release* preferibilmente) e magari il codice sorgente .cs se open-source. Indica sempre i requisiti (ad es. "Richiede ScriptHookV e ScriptHookV .NET v3.6 o superiore, e magari altre librerie come **NativeUI** se le usi per menu UI). Gli utenti dovranno solo piazzare la tua DLL nella loro cartella scripts.

# 6. Risorse utili per approfondire

Ora che hai le basi, ci sono tantissime risorse online per aiutarti a migliorare e creare script più complessi. Eccone alcune consigliate:

- **Documentazione Ufficiale di ScriptHookV .NET:** il progetto SHVDN è open source e ha una Wiki su GitHub con guide e **FAQ**. Ad esempio, la pagina *Getting Started* e *How-Tos* forniscono esempi pratici (molti dei quali ripresi in questa guida) (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Inoltre, è disponibile una reference completa delle classi e metodi sul sito (auto-generato) **nitanmarcel.github.io/scripthookvdotnet/**, dove puoi vedere tutti i membri del namespace GTA, GTA.Native, ecc., con spiegazioni. Per esempio, puoi trovarci l'elenco dei **Native Functions** e come chiamarli (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub).

- **Community e forum:** La comunità di modding è attiva. Il forum di riferimento è la sezione *Coding* su **GTAForums.com**, dove c'è un thread dedicato a ScriptHookV .NET e molti topic con domande e codice di esempio. Anche su Reddit ci sono discussioni (es. subreddit r/GTAV_Mods). Se incontri problemi, cerca su questi forum: spesso la domanda è già stata fatta da altri. Ad esempio, trovi discussioni su *come intercettare input* o *differenze ScriptHookV vs RPH* (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods), su come risolvere errori comuni, ecc.
    - Un forum italiano specifico non è molto attivo su questi temi, ma puoi comunque confrontarti con altri italiani su community generali di gaming modding.
- **Tutorial video:** Se preferisci vedere in pratica i passaggi, ci sono ottime serie su YouTube. Ad esempio:
    - *GTA V Scripting Tutorial* (serie in più parti, in inglese, di **Metiri**): parte 1 copre installazione e setup, parte 2 la creazione del primo script, parte 3 e successive introducono funzionalità più avanzate (menu, ecc.) – ottimo per seguire passo passo.
    - *How To Make a GTA 5 Mod (C#)* di **AbstractMode** (serie in inglese): mostra dalla configurazione all'implementazione di mod semplici, con esempi pratici.
    - Assicurati che i video non siano troppo datati: cerca tutorial dal 2019 in poi, così le informazioni sono aggiornate (tenendo conto di ScriptHookV .NET 3.x).
- **Altre librerie utili:** man mano che cresci nello scripting, potresti voler aggiungere funzionalità avanzate come interfacce utente menu, grafica 2D/3D, ecc. Due risorse popolari:
    - **NativeUI** – una libreria .NET (sviluppata originariamente da Guadmaz) per creare menu in stile GTA (elenchi, item selezionabili) facilmente. Molti mod usano NativeUI per offrire un menu di configurazione in-game.
    - **LemonUI** – è un progetto più recente che ha sostituito NativeUI (è simile concettualmente, ma più aggiornato e supporta anche Red Dead 2). Puoi usarlo per creare menu più complessi con meno sforzo.
    - Queste librerie sono da includere come .dll aggiuntive nel progetto e nella cartella scripts. Documentati sui rispettivi repository per come usarle.
- **Database delle funzioni native (Native DB):** Per sfruttare al massimo GTA V, a volte dovrai chiamare funzioni native non esposte direttamente dall'API .NET. Il sito **alloc8or (Native DB)** contiene la lista completa delle native di GTA V con nomi, hash e parametri. È uno strumento indispensabile quando userai `Function.Call` di GTA.Native (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Esempio: vuoi far comparire un effetto grafico o cambiare una stat del giocatore non prevista dalle classi GTA? Trovi la funzione su Native DB e la invochi. (Ricorda di includere `GTA.Native` e di usare l'enum `Hash` se disponibile, oppure l'hash a 64bit della funzione).

Con queste risorse e tanta pratica, potrai creare mod sempre più sofisticati – dal semplice trainer personalizzato, a missioni scriptate, fino a sistemi completi come mini-giochi dentro GTA V. Ricorda di procedere per gradi, sperimentare e consultare la community quando sei bloccato. Buon modding e buon divertimento con i tuoi script per Los Santos!

**Fonti utilizzate:**

- Documentazione ufficiale e wiki di ScriptHookV .NET (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub)
- Tutorial *Introduction to Scripting – GTA5 Photography* (The Photographer's Guide to Los Santos) (The Photographer's Guide to Los Santos)
- Esempi dal repository ScriptHookV .NET (GitHub) (Getting Started · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub)
- Discussioni comunitarie (GTAForums, Reddit) su differenze tra ScriptHookV, SHVDN e RPH (ScriptHookV, ScriptHookV .Net & RagePluginHook - Which hook to use? : r/GTAV_Mods)
- Pagina ufficiale di download ScriptHookV .NET (gta5-mods.com) ( Community Script Hook V .NET - GTA5-Mods.com )

Perfetto, avvierò subito una ricerca approfondita per raccogliere guide, esempi di codice, mod esistenti e best practices per l'implementazione di poteri sovrumani in GTA V utilizzando ScriptHookVDotNet3 in C#, compatibili con il progetto MarvelScript. Ti fornirò tutto il materiale in italiano, strutturato e spiegato in dettaglio direttamente qui in chat non appena sarà pronto.

# Implementare poteri sovrumani in GTA V con ScriptHookVDotNet3 (C#)

Implementare poteri **stile supereroe** in GTA V richiede di sfruttare **ScriptHookV + ScriptHookVDotNet3**, che permettono di scrivere script in C#. Bisogna conoscere la struttura base di uno script .NET per GTA V, l'uso dei **metodi e proprietà dell'API di ScriptHookVDotNet**, e le **funzioni native** di GTA (tramite `Function.Call`) per effetti avanzati. Di seguito presentiamo guide e risorse, esempi di codice per vari poteri (volo, velocità, forza, ecc.), best practices per stabilità e design modulare, e link utili (mod esistenti, discussioni e tutorial). *Nota:* Tutto il materiale è aggiornato e compatibile con **ScriptHookVDotNet v3** e GTA V single-player (non online).

## Guide e tutorial per poteri personalizzati con ScriptHookVDotNet3

**Struttura di uno script e gestione input:** Uno script .NET in GTA V deriva dalla classe base `Script` e tipicamente utilizza gli eventi `Tick`, `KeyUp` e `KeyDown` per aggiornamenti continui e input da tastiera (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ) (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ). Ad esempio, possiamo intercettare un tasto per attivare un potere e usare `Tick` per applicarne gli effetti ogni frame. Il frammento seguente mostra la struttura minima di uno script SHVDN3 (v3) che spawna un veicolo quando si preme *NumPad1* (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ):

```
public class CreateVehicle : Script {
    public CreateVehicle() {
        Tick += OnTick;
        KeyUp += OnKeyUp;
    }
    private void OnTick(object sender, EventArgs e) { }
    private void OnKeyUp(object sender, KeyEventArgs e) {
        if (e.KeyCode == Keys.NumPad1) {
            Vehicle veh = World.CreateVehicle(VehicleHash.Zentorno,
                        Game.Player.Character.Position +
Game.Player.Character.ForwardVector * 3.0f);
            UI.Notify(\"Veicolo creato!\");
        }
    }
}
```

In questo esempio, `Game.Player.Character` rappresenta il **Ped del giocatore** (il nostro eroe) e viene usato per ottenere posizione e direzione (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ). **Concetti chiave**: usare `World` per creare oggetti (veicoli, ped, effetti), `Game.Player.Character` per manipolare il protagonista, e `UI.Notify` per messaggi di debug a schermo.

**Utilizzo di funzioni native con `Function.Call`:** ScriptHookVDotNet espone molte funzioni dell'engine, ma per alcune operazioni avanzate potrebbe servire invocare direttamente i **nativi di GTA V**. In SHVDN, ciò avviene con `GTA.Native.Function.Call(Hash.<NOME_FUNZIONE>, param1, param2, ...)`. Ad esempio, l'API non ha una proprietà dedicata per la radio veicolo, ma possiamo disattivarla col nativo `SET_VEHICLE_RADIO_ENABLED` usando il relativo hash (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Per scoprire hash e parametri dei nativi esiste la **Native DB** di alloc8or (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). In generale, molti poteri speciali (es. effetti particellari, modifica forze, ecc.) richiedono `Function.Call` su nativi appropriati.

**Creazione di abilità come volo, forza, laser, invisibilità:** Non esiste una "funzione magia" per ogni superpotere – vanno combinati elementi base:

- *Volo*: si può implementare **disabilitando la gravità del ped** e controllandone manualmente la posizione o velocità. Ad esempio, ScriptHookVDotNet fornisce la proprietà `Entity.HasGravity` per gli oggetti (Ped eredita da Entity) (Class Ped | Community Script Hook V .NET ). Impostando `Game.Player.Character.HasGravity = false` il personaggio smette di cadere. Nel loop `Tick` possiamo poi applicare forze in direzione avanti/indietro in base all'input. La direzione del ped è data da `Entity.ForwardVector` (Class Ped | Community Script Hook V .NET ), mentre la funzione `Entity.ApplyForce()` permette di spingerlo (Method ApplyForce | Community Script Hook V .NET ). Un approccio comune è: quando il *volo* è attivo, se il giocatore preme W o S si usa `ApplyForce` in avanti/indietro, e un altro tasto (es. Shift/Ctrl) per salire o scendere ( Psychokinetic [W.I.P] - GTA5-Mods.com ). Questo simula un volo controllato. Inoltre, si può impostare `Ped.Task.ClearAll()` per interrompere animazioni (come il paracadute) e usare animazioni personalizzate di volo se disponibili. Il mod **Superman** di JulioNIB conferma questo schema: si attiva/disattiva il volo con un tasto (es. *Space* in aria) e si controlla con mouse e WASD, aumentando la velocità in base all'input (GTA X Scripting - JulioNIB mods: GTA V Superman script mod) (GTA X Scripting - JulioNIB mods: GTA V Superman script mod). Durante il volo conviene anche rendere il ped invincibile e senza collisioni (per non sbattere continuamente): proprietà come `Entity.IsInvincible = true` (Class Ped | Community Script Hook V .NET ) e magari `Entity.HasCollision = false` possono essere utili.
- *Super velocità*: GTA V limita la velocità di corsa tramite un moltiplicatore massimo (~1.49×) (nbase - smth like "nativedb" - RAGE Multiplayer). Per aumentare leggermente la velocità, si può usare il nativo **SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER** sul player (nbase - smth like "nativedb" - RAGE Multiplayer). Esempio: `Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER,`

`Game.Player, 1.4f)` aumenta la velocità di corsa del 40%. Tuttavia, per velocità **Flash** estreme, serve un approccio personalizzato: i mod avanzati disabilitano il limite di 1.49 sfruttando stratagemmi come applicare forze impulsive in avanti ogni tick quando il ped corre, oppure manipolare la posizione incrementandola manualmente (teletrasporto rapido). Ad esempio, la mod *The Flash* v2 di JulioNIB permette tre livelli di velocità configurabili nel .ini (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle) e gestisce **scia visiva** e slow-motion automatico per dare l'effetto cinematografico. In pratica, per **far correre un ped più veloce**, possiamo: (a) aumentare il multiplier di sprint al massimo consentito quando il potere è attivo, (b) ogni tick se il ped **IsRunning** o si muove avanti, applicare una spinta aggiuntiva: `Game.Player.Character.ApplyForce(Game.Player.Character.ForwardVector * forzaExtra)`, e (c) gestire eventuali effetti visivi (es. motion blur, scie di particelle) per enfatizzare la super velocità (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle). Un dettaglio implementativo: con velocità altissime si rischia di oltrepassare gli oggetti prima che la fisica li rilevi; per mitigare, i mod attivano una leggera *slow-motion* in automatico quando si supera una certa velocità (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle) (rallentando il tempo di gioco, le collisioni funzionano meglio).

- *Super salto*: GTA V include un cheat "Super Jump". In SHVDN esiste il metodo `Game.Player.SetSuperJumpThisFrame()` che abilita il super salto per il giocatore **nel frame corrente** (Starman/Main.cs at master · theandrew61/Starman · GitHub). Si può chiamare ad ogni tick quando un flag "super salto" è attivo, oppure intercettare il momento del salto e applicare una forza verso l'alto. Un esempio: lo script open-source **Starman** abilita il super jump se l'opzione è attiva, usando proprio `player.SetSuperJumpThisFrame()` nel suo OnTick (Starman/Main.cs at master · theandrew61/Starman · GitHub). In alternativa, per maggiore controllo, si può rilevare quando il giocatore preme il tasto salto (space) e se un potere di mega-salto è attivo, allora applicare manualmente: `Game.Player.Character.Velocity = new Vector3(0, 0, 20f);` (dando un impulso verso l'alto). **Nota:** impostare direttamente `Velocity` funziona istantaneamente (Class Ped | Community Script Hook V .NET ), mentre `ApplyForce` con un vettore Z positivo può dare un effetto più graduale. Julionib nei suoi mod Hulk/Superman combina entrambe le cose per enormi balzi.
- *Forza e danni sovrumani*: Per aumentare la **forza fisica** del personaggio (es. pugni che mandano i nemici in volo, capacità di spostare veicoli), possiamo agire su vari fronti:
    - Aumentare i **danni del melee**: GTA V non ha un parametro semplice per la forza del pugno, ma si possono utilizzare nativi come `SET_PED_MELEE_MOVEMENT_CLIPSET` o direttamente applicare una forza all'entità colpita quando avviene un impatto. Un trucco comune: intercettare l'evento di attacco (non trivialissimo senza librerie esterne) oppure verificare, ad ogni tick, quali ped vicini sono stati colpiti (controllando la loro salute o se sono a terra) e applicare su di loro `p.Kill()` o `p.ApplyForce` per farli volare via.
    - **Lanciare oggetti/persone**: se l'eroe può afferrare veicoli o ped e scagliarli, si può usare `Entity.AttachTo()` per attaccare un'entità alla mano del giocatore, e poi

`Detach` applicando una forza. Ad esempio, per sollevare un'auto: `vehicle.AttachTo(player, player.GetBoneIndex(Bone.SKEL_R_Finger01), offset, rotation);` la "incolla" alla mano; quando si preme il tasto per lanciare, si fa `vehicle.Detach()` e subito `vehicle.ApplyForce(player.ForwardVector * forzaLancio + Vector3.WorldUp * forzaVerticale)`. Così Hulk può scagliare auto lontano.

- **Massa e resistenza**: Nel mod Superman, JulioNIB aumenta la "massa" del personaggio, così se viene investito da un'auto, è l'auto a subire danni e il ped rimane fermo (GTA X Scripting - JulioNIB mods: GTA V Superman script mod). Non c'è un metodo diretto per la massa del ped, ma l'effetto si ottiene rendendo il ped invincibile e applicando danno inverso al veicolo. Ad esempio, nel *Tick*, per ogni veicolo vicino colliso con il giocatore (`vehicle.IsTouching(Game.Player.Character)`), si può fare `vehicle.EngineHealth -= 50f` o usare un'esplosione leggera per danneggiarlo. Il risultato percepito è che il nostro eroe è **inamovibile** e le auto si distruggono urtandolo (GTA X Scripting - JulioNIB mods: GTA V Superman script mod). Inoltre, attivare flag di invulnerabilità come `Ped.IsFireProof`, `IsExplosionProof`, etc., protegge da danni ambientali (Class Ped | Community Script Hook V .NET ).

- *Laser dagli occhi / raggi energetici*: Questo richiede **effetti particellari** e applicazione di danno a distanza. Un modo è sfruttare i **Particle FX** di GTA: ad esempio, GTA V ha effetti come il raggio alieno (`scr_rcbarry1` / `scr_alien_teleport`) utilizzati nelle missioni. Si può riprodurre un effetto di laser usando i nativi di particle effects. SHVDN v3 consente di caricare e usare effetti con `Function.Call` sui nativi come `REQUEST_NAMED_PTFX_ASSET` e `START_PARTICLE_FX_LOOPED_ON_ENTITY`. Un esempio pratico viene ancora dallo script Starman: per l'effetto scintillante del power-up, ogni 0.5 secondi esegue (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub):

  ```
  ParticleOnEntity(player.Character, "scr_rcbarry1", "scr_alien_teleport", 1.0f);
  ```

  *(Qui `ParticleOnEntity` è una funzione custom che richiama il nativo per avviare l'effetto `scr_alien_teleport` sull'entità indicata con un determinato scale.)* Dunque, per un **raggio laser** si potrebbe attaccare un effetto particellare continuo all'origine (gli occhi del ped, ottenendo le coordinate di un bone della testa) e un secondo effetto all'impatto sul target. Per colpire i nemici, si può fare un **raycast** in direzione dello sguardo del player (`World.Raycast` dal ped in avanti) e se colpisce un Ped o Veicolo, applicare danno: ad esempio `targetPed.Health -= 50` o usare un'esplosione invisibile (AddExplosion - FiveM Natives @ Cfx.re Docs). JulioNIB nel suo mod Superman implementa **laser occhi** che bruciano i bersagli e fanno esplodere le gomme (GTA X Scripting - JulioNIB mods: GTA V Superman script mod) (GTA X Scripting - JulioNIB mods: GTA V Superman script mod), segno che usava un effetto fuoco e poi `tyreBurst` sul veicolo colpito. Un ulteriore abbellimento: aggiungere suono (utilizzando

`Function.Call(Hash.PLAY_SOUND_FRONTEND, ...)` oppure tramite NAudio come fa Starman per suonare la musica di invincibilità (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub)).

- *Invisibilità*: Si può rendere il **player invisibile** impostando `Game.Player.Character.IsVisible = false` (Class Ped | Community Script Hook V .NET ). Questo fa sì che il modello 3D scompaia, ma attenzione: gli NPC continueranno a individuarlo visivamente (perché quel flag non incide sull'AI). Per un effetto "stealth", bisognerebbe anche impostare il ped in modalità furtiva o modificare le relazioni (es. far sì che i nemici lo ignorino finché non attacca). Un metodo semplice per evitare l'aggressione immediata è usare `Ped.SetInvisible()` e anche `Ped.IsCollisionProof = true` (Class Ped | Community Script Hook V .NET ) (così nessuno lo urta). Però *ai fini visivi* basta IsVisible. Alcuni mod combinano anche un **effetto di rifrazione**: ad es. spawnano un piccolo **particellare trasparente** addosso al ped per simulare la distorsione (facoltativo).
- *Teletrasporto*: Teletrasportare il giocatore è piuttosto semplice: basta assegnare una nuova posizione al ped: `Game.Player.Character.Position = new Vector3(x, y, z)` (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). La guida ufficiale mostra che funziona sia per player che per altre entità. Un esempio: per teletrasportare alla meta del GPS, si può ottenere la posizione del waypoint (se esiste) e settarla. Attenzione a recuperare la coordinata Z corretta del suolo: spesso si usa il nativo `GET_GROUND_Z_FOR_3D_COORD` per trovare l'altezza del terreno (GetGroundZ is error prone · Issue #180 · scripthookvdotnet ... - GitHub). Un teletrasporto "soft" (con effetto) può usare fade out/in dello schermo: chiamare `Function.Call(Hash.DO_SCREEN_FADE_OUT, durata)` prima del TP e `...FADE_IN` dopo. Un tutorial video mostra la creazione di uno script di teletrasporto con ScriptHookV .NET passo-passo (GTA:V ScriptHookV Coding: Part Three | Teleport Player ... - YouTube). In alternativa al set di posizione, c'è anche un nativo `SET_ENTITY_COORDS` equivalente. La mod **Psychokinetic** include la possibilità di teletrasportarsi dove si guarda (tasto mouse centrale) ( Psychokinetic [W.I.P] - GTA5-Mods.com ) ( Psychokinetic [W.I.P] - GTA5-Mods.com ): presumibilmente usando un raycast in direzione della visuale e poi posizionando il player lì.

**Gestione di particelle, audio e altri effetti:** Abbiamo accennato ai **Particle FX**. Esistono risorse che elencano i nomi degli effetti (es. sul forum di GTA5-Mods alcuni script come *Particle Effects [.NET]* ( Particle Effects - GTA5-Mods.com ) ( Particle Effects - GTA5-Mods.com )). Per riprodurre un effetto:

1. Caricare l'asset con `Function.Call(Hash.REQUEST_NAMED_PTFX_ASSET, "nomeAsset")` (es. *scr_rcbarry1.ypt*).
2. Impostare l'asset per la prossima chiamata: `Function.Call(Hash._SET_PTFX_ASSET_NEXT_CALL, "nomeAsset")`.
3. Avviare l'effetto con `Function.Call(Hash.START_PARTICLE_FX_LOOPED_ON_ENTITY, "nomeFx", ent.Handle, offsetX, offsetY, offsetZ, rotX, rotY,`

`rotZ, scale, 0, 0, 0`). È un po' macchinoso, ma SHVDN semplifica con *overload* `World.CreateParticleEffectLooped(...)` (a seconda della versione).

4. Salvare eventualmente l'handle dell'effetto se vogliamo interromperlo poi con `REMOVE_PARTICLE_FX`.

Per **audio**, ScriptHookVDotNet offre metodi come `GTA.Audio.SetAudioFlag` o la possibilità di riprodurre suoni frontend. Se servono suoni personalizzati (es. effetti sonori di Iron Man), bisogna utilizzare librerie esterne (es. NAudio, come incluso nello script Starman (GitHub - theandrew61/Starman: A mod for GTA V that grants you invincibility like in Mario games.) (GitHub - theandrew61/Starman: A mod for GTA V that grants you invincibility like in Mario games.) per la musica di invincibilità). In molti casi però si possono riutilizzare suoni di GTA (ci sono nativi per suoni di esplosioni, armi, ecc.).

**Danni, forze, ragdoll e fisica:** L'API .NET espone proprietà utili: `Entity.Health` e `Entity.ApplyDamage(int damage)` per gestire i danni direttamente. Ad esempio, per **ferire un ped vicino**: `enemy.Health -= 50; if(enemy.Health <= 0) enemy.Kill();`. Per esplosioni, c'è `World.AddExplosion(position, ExplosionType.Grenade, damageScale, radius)` (Class World | Community Script Hook V .NET). Anche l'**uso di esplosioni invisibili** è una tattica per simulare colpi potenti: settando i parametri *isAudible=false*, *isInvisible=true* il gioco applica danno fisico e spostamento senza effetti visivi né suono (AddExplosion - FiveM Natives @ Cfx.re Docs). Questo è utile per poteri come colpi di Hulk a terra o onde d'urto – in pratica chiami un'esplosione invisibile alla posizione giusta per buttar via oggetti attorno.

Il **ragdoll** dei ped (modalità pupazzo) può essere attivato/disattivato: `Ped.CanRagdoll = false` rende il nostro eroe stabile (es. Superman non cade mai a terra se investito) (Starman/Main.cs at master · theandrew61/Starman · GitHub). Viceversa, lanciare i nemici in ragdoll: `targetPed.Ragdoll(duration)` li fa stramazzare per un tot di millisecondi. La mod Starman ad esempio disabilita il ragdoll del player durante l'effetto invincibilità (Starman/Main.cs at master · theandrew61/Starman · GitHub). Anche parametri come `Ped.CanPlayAmbientAnims` o `Ped.StaysInVehicleOnJack` possono essere regolati per evitare che certe azioni interrompano i poteri.

**Controllo dell'IA e interazioni dinamiche:** Per far sì che i **nemici o alleati con poteri** si comportino adeguatamente, dobbiamo agire su AI e **tasks**:

- **Trovare entità vicine:** `World.GetNearbyPeds(position, raggio)` ritorna un array di Ped entro un certo raggio (Starman/Main.cs at master · theandrew61/Starman · GitHub). Possiamo usarlo per trovare NPC da colpire o salvare. Lo snippet di Starman, ad esempio, cerca ped entro un raggio di "distruzione" e li uccide immediatamente quando il power-up è attivo (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub) (simulando un'aura mortale).
- **Task e comportamenti:** SHVDN espone la proprietà `Ped.Task` (o `Ped.Tasks`) che permette di assegnare compiti: ad esempio `enemy.Tasks.FightAgainst(Game.Player.Character)` fa attaccare il giocatore, oppure `ally.Tasks.FollowToOffsetFromEntity(Game.Player.Character,`

`offset, timeout)` per far seguire. Nel contesto di **villain vs eroe**, si può creare un Ped con stessi poteri e aggiornarlo in Tick: farlo correre veloce come il player, attivare poteri periodicamente (lanci di oggetti, laser, ecc.). Un aspetto cruciale è settare le **relazioni**: GTA V ha gruppi e relazioni (es. poliziotti contro player). Possiamo creare un nuovo *RelationshipGroup* per supereroi e un altro per nemici e impostare relazioni ostili tra loro usando `World.AddRelationshipGroup` e `Ped.RelationshipGroup`. Così, un *villain* attaccherà l'eroe e viceversa automaticamente (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle).

- **Eroe controllabile vs ped controllati:** Per *mind control* (controllo mentale) potremmo o *trasferire* il controllo del giocatore a un NPC (usando `Game.Player.Character = ped`, che però richiede ChangeModel e può essere complicato), oppure più semplicemente **scriptare le azioni del ped** come se fosse controllato. Ad esempio, se il nostro eroe è Professor X e vuole far combattere un ped dalla sua parte, possiamo prendere un ped ostile, cambiare la sua fazione a neutrale o ally rispetto al player (`targetPed.RelationshipGroup = player.RelationshipGroup`), e poi assegnargli un task di attaccare gli altri nemici rimasti. In pratica diventa un *ally temporaneo*.
- **Spawn di NPC con poteri**: JulioNIB nei suoi mod consente di spawnare "*nemico/alleato flash*" nel mod Flash (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle) e Superman nemico/alleato in altri mod. In Flash v2, quando un enemy Flash è attivo, il gioco disabilita l'auto-slowmotion per garantire che anche l'AI si muova veloce (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle). Questo significa che i tick di movimento del ped nemico venivano probabilmente gestiti con la stessa logica del player (stessi poteri applicati). Un trucco è avere una classe Power che accetta un Ped su cui agire (così lo stesso codice di volo, velocità ecc. può essere applicato sia al giocatore sia a un NPC, semplicemente passando il Ped appropriato invece di `Game.Player.Character`). In pseudocodice:

```
if (ped != Game.Player.Character) {
    // AI villain logic
    ped.Tasks.ChaseWithMotion(Game.Player.Character); // ipotetico
    // applica poteri sul ped: es. velocità aumentata
    ped.MoveSpeed = boostedSpeed;
}
```

Naturalmente, questo richiede calcoli in Tick per ogni ped controllato, il che aumenta la complessità (e attenzione alle performance se i ped con poteri sono molti).

# Esempi di codice per poteri specifici

Di seguito alcuni **snippet/commenti** per implementare specifici poteri e funzionalità:

- **Volo controllato:** Abilitare/disabilitare la gravità e applicare forze manuali. Esempio semplificato:

```
// Attiva volo
playerPed.HasGravity = false;
playerPed.IsInvincible = true;
// In Tick:
Vector3 vel = playerPed.Velocity;
if (Game.IsKeyPressed(Keys.W)) {
```

```
    // spinta in avanti
    playerPed.Velocity = playerPed.ForwardVector * 20f;
}
if (Game.IsKeyPressed(Keys.S)) {
    // spinta indietro (rallenta/inverte)
    playerPed.Velocity = playerPed.ForwardVector * -10f;
}
if (Game.IsKeyPressed(Keys.ShiftKey)) {
    // sali in quota
    playerPed.Position += new Vector3(0, 0, 2f);
}
```

Questo codice prenderebbe input di base. Nella pratica, useremo `ApplyForce` per morbidezza, e un controllo del mouse per orientare il ped in volo (magari sincronizzandolo con la direzione della telecamera). Il mod **Psychokinetic** offre 3 modalità di volo (cambiabili con *CapsLock*) e tasti Shift/Ctrl per salire/scendere ( Psychokinetic [W.I.P] - GTA5-Mods.com ), confermando questa logica. Per animazione, possiamo forzare un anim loop di nuoto o di caduta stile volo.

- **Super velocità e "flash time":** Per implementare Flash, potremmo usare qualcosa del genere:

```
// Attiva supervelocità
Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER, Game.Player,
1.49f);
// E magari anche aumentare temporaneamente la velocità di animazione:
Function.Call(Hash.SET_TIME_SCALE, 0.5f); // slow-motion se volessimo
// In Tick:
if (playerPed.IsRunning) {
    // applica spinta extra in avanti
    playerPed.ApplyForce(playerPed.ForwardVector * 5.0f);
    // genera effetto scia
    World.AddExplosion(playerPed.Position - playerPed.ForwardVector*2,
ExplosionType.Smoke, 0.0f, 0.1f, false, true);
}
```

(L'esplosione di tipo Smoke invisibile crea solo un po' di fumo come scia). Una *vera* scia si fa con particelle di polvere a terra o un effetto di motion blur – quest'ultimo non è direttamente accessibile, ma mods come Flash usano texture e shader custom. Comunque, l'esempio illustra l'uso del multiplier nativo e forze addizionali.

- **Salti enormi:** Usare il super jump integrato:

```
if (powerSuperJump && Game.IsKeyPressed(Keys.Space)) {
    Game.Player.SetSuperJumpThisFrame();
}
```

Così, ogni frame in cui il potere è attivo e il giocatore preme *salto*, GTA applicherà un salto molto più alto del normale (come fa il cheat code) (Starman/Main.cs at master · theandrew61/Starman · GitHub). Il codice di Starman dimostra l'utilizzo di `SetSuperJumpThisFrame` con un flag dalle impostazioni (Starman/Main.cs at master · theandrew61/Starman · GitHub). In alternativa manuale:

```
if (Game.IsKeyPressed(Keys.Space) && playerPed.IsOnGround) {
    playerPed.ApplyForce(Vector3.WorldUp * 6.0f, Vector3.Zero,
ForceType.MaxForceRot);
}
```

dove `ForceType.MaxForceRot` indica di applicare una spinta forte ignorando i limiti normali.

- **Lancio ragnatela / rampino (tipo Spider-Man):** Si può implementare analogamente al **grappling hook** di Just Cause. GTA V ha nativi per creare corde fisiche: `ADD_ROPE(...)` e `ATTACH_ENTITIES_TO_ROPE(rope, ent1, ent2, ...)`. JulioNIB nel suo mod *Just Cause Grappling Hook* permetteva di **attaccare il rampino a edifici, ped e veicoli**, anche multipli, e "tirare" il giocatore verso il punto o l'oggetto verso il giocatore (GTA X Scripting - JulioNIB mods: GTA V - Just Cause 2 Grappling hook mod) (GTA X Scripting - JulioNIB mods: GTA V - Just Cause 2 Grappling hook mod). Un esempio di logica:
  - Quando il giocatore spara la ragnatela (es. tasto destro+sinistro mirato su un veicolo), si lancia un raycast; se colpisce un'entità valida, creare una corda: `ropeId = Function.Call<int>(Hash.ADD_ROPE, startPos.X, startPos.Y, startPos.Z, 0,0,0, length, RopeType.ThickRope, length, 0.5, false, false, false, 0.0f);`.
  - Attaccare il rope all'entità target e al giocatore (o a un punto fisso della mappa): `Function.Call(Hash.ATTACH_ENTITIES_TO_ROPE, ropeId, playerPed, targetEnt, playerBoneIndex, 0,0,0, targetOffsetX,... )`.
  - Eventualmente usare `APPLY_FORCE` per tirare: ad esempio per *tirarsi verso* il bersaglio, applicare forza al player verso la posizione del bersaglio; per *tirare l'oggetto*, applicare forza al target verso il player.
  - Permettere di **recuperare/rilasciare** la corda: `DELETE_ROPE(ropeId)` o utilizzare `START_ROPE_UNWINDING/WINDING`.

  Se questa implementazione è complessa, un'alternativa è simulare la ragnatela con un semplice raggio invisibile: in ogni tick, se il target è agganciato, settiamo `target.Position` gradualmente verso il player (per tirarlo) o `player.Position` verso il target (per oscillarsi). Non c'è fisica realistica ma è più facile. Ad esempio, per dondolarsi come Spider-Man tra palazzi, il mod **Grappling Hook** effettua un attacco a parete e poi applica movimento pendolare (Julio menziona la possibilità di attaccarsi ai muri e usare il paracadute come sostituto di oscillazione) (Just Cause 3 script mod - Grappling hook for GTA 5) (Just Cause 3 script mod - Grappling hook for GTA 5).

  Codice semplificato per tirare un ped verso di noi:

```
Ped target = ... // ped colpito da ragnatela
if (target != null) {
    Vector3 dir = playerPed.Position - target.Position;
    target.ApplyForce(dir.Normalized * 10f);
}
```

  E viceversa per tirare noi verso un punto:

```
Vector3 dir = hookPoint - playerPed.Position;
playerPed.ApplyForce(dir.Normalized * 15f);
```

Infine, per rilasciare un ped legato e farlo volare via: `target.Detach(); target.Velocity = playerPed.ForwardVector * 20f + Vector3.WorldUp * 5f;`. Questo lo scaglia in aria. La mod *JC3 Grappling* include inoltre **riavvolgimento del cavo** (tasto X) (GTA X Scripting - JulioNIB mods: GTA V - Just Cause 2 Grappling hook mod) e possibilità di attaccare più entità insieme (collegare due veicoli e poi accorciare la corda).

- **Esplosioni controllate e poteri distruttivi:** Per un potere tipo *colpo energetico* o *esplosione gamma*, possiamo usare `World.AddExplosion` dove e quando serve. Esempio: un colpo di Hulk a terra:

```
Vector3 pos = playerPed.Position;
World.AddExplosion(pos, ExplosionType.Grenade, 5.0f, 1.0f, true, false);
```

  Questo genera un'esplosione tipo granata con danno 5.0 e raggio 1.0 (locale, tipo shockwave) (Class World | Community Script Hook V .NET). Impostando `isAudible=true, isInvisible=false` avremo suono e fuoco. Per un effetto *onda d'urto senza fuoco*, usare `ExplosionType.SoundGrenade` e `isInvisible=true`. Alcuni mod sfruttano esplosioni per implementare **colpi terra-terra**: il mod Iron Man, ad esempio, quando atterra crea un impatto che spacca il marciapiede (GTA 5 Iron Man script mod launches awesome Version 2.0 | PC Gamer) (nuova animazione di landing heavy). Anche i **fulmini di Storm/Thor** possono essere simulati: un fulmine è essenzialmente un effetto visivo (c'è un nativo specifico `ADD_LIGHTNING` in ambienti, ma non molto controllabile) combinato con un'esplosione elettrica su un target.

- **Controllo mentale (NPC):** come discusso, la strada è manipolare l'AI. Un esempio di *mind control* su un ped vicino potrebbe essere:

```
Ped enemy = World.GetNearbyPeds(playerPed, 30f).FirstOrDefault(p =>
p.IsHuman && p.IsHostile);
if(enemy != null) {
    enemy.Task.ClearAll();
    enemy.RelationshipGroup = Game.Player.Character.RelationshipGroup;
    enemy.Task.FightAgainstHatedTargets(50f);
    UI.Notify($"{enemy} ora combatte per te!");
}
```

  Questo frammento ipotetico prende un nemico, lo *sgancia* dai suoi compiti attuali, lo assegna al gruppo del player (diventando amico) e poi gli fa attaccare i suoi bersagli odiati (che prima erano alleati e ora diventano nemici ai suoi occhi). Bisogna anche considerare di dargli qualche "poteretto" se vogliamo che sia un superNPC – ad esempio aumentargli la vita o armi. In alternativa estrema, si potrebbe **scambiare il controllo**: salvare lo stato del nostro eroe, trasformare il player nel ped bersaglio (`Game.Player.ChangeModel(enemy.Model)` crea un ped giocante con quel modello (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub)), ma questo è complicato e spesso non necessario.

- **Trasformazioni e cambio personaggio:** Molti supereroi *si trasformano* (es. Bruce Banner in Hulk, o armatura di Iron Man che si equipaggia). Con SHVDN possiamo cambiare il modello 3D del player con `Game.Player.ChangeModel()` come visto sopra (How Tos

· scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Ad esempio, per trasformarsi in Hulk:

```
Model hulkModel = new Model("HULK"); // se esiste un ped model aggiunto
chiamato HULK
hulkModel.Request(500);
while (!hulkModel.IsLoaded) Script.Wait(100);
Game.Player.ChangeModel(hulkModel);
Game.Player.Character.Style.SetDefaultClothes();
hulkModel.MarkAsNoLongerNeeded();
```

Il codice carica il model (deve essere installato come ped aggiuntivo o sostituzione), aspetta il caricamento, poi esegue ChangeModel (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). `SetDefaultClothes()` sistemerà gli outfit di base. Si può fare il processo inverso per tornare umano. Attenzione: cambiando model si perdono riferimenti al ped precedente (ad es. se avevamo attaccato oggetti a Bruce Banner, vanno riattaccati al nuovo ped Hulk). Per effetti visivi di trasformazione, si può riprodurre un'animazione (se disponibile) o particelle (es. un fumo che copre il cambio).

- **HUD e barre personalizzate:** Molti mod di superpoteri aggiungono interfacce (HUD) per stamina, carica potere, selettore di poteri, ecc. ScriptHookVDotNet offre alcune funzioni di base per il disegno 2D – ad esempio `UI.DrawTexture` o la possibilità di usare la libreria **NativeUI** (che permette di creare menu e alcune componenti). Ad esempio, nel mod Starman viene usata una **Barra di durata** del potere invincibilità: crea un oggetto `BarTimerBar` denominato "STARMAN POWER" (Starman/Main.cs at master · theandrew61/Starman · GitHub), e ad ogni tick aggiorna `btb.Percentage` in base al tempo rimasto (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub), poi chiama `tbPool.Draw()` per disegnarla sullo schermo (Starman/Main.cs at master · theandrew61/Starman · GitHub). Questo è simile alle barre che compaiono in GTA nelle missioni (in basso a destra). Per utilizzarle bisogna includere il namespace `GTA.UI` o utilizzare NativeUI extras. In alternativa, si può disegnare testo e forme con `UI.DrawRect` e `UI.DrawText`. Anche la *ruota dei poteri* (power wheel) come in mod Flash o Psychokinetic si può implementare intercettando un tasto (es. Q) per aprire un menu circolare. La mod **Psychokinetic** ne ha una molto elaborata con mouse/analogico per selezione ( Psychokinetic [W.I.P] - GTA5-Mods.com ) ( Psychokinetic [W.I.P] - GTA5-Mods.com ). Probabilmente utilizzano texture semi-trasparenti e calcoli trigonometrici per le sezioni della ruota.
- **Cooldown e limitazioni temporali:** Per evitare spam di un potere, si impone un tempo di recupero. Un modo semplice: tenere un `DateTime lastUsed` per ogni potere. Quando il giocatore tenta di riusarlo, controllare se `DateTime.Now - lastUsed >= cooldownDuration`. In caso contrario, magari mostrare un messaggio "Potere in ricarica...". Nel codice Starman, dopo la disattivazione del potere invincibile, viene impostato un cooldown random 1-5 secondi durante i quali se si ripreme il tasto si mostra un Notify di attesa (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub). Un altro approccio è usare una **variabile contatore** decrementata in Tick. Ad esempio:

```
int teleportCooldown = 0;
...
if (Game.IsKeyPressed(Keys.T) && teleportCooldown <= 0) {
    TeleportTo(...);
    teleportCooldown = 300; // 300 ticks di cooldown (~5 secondi se 60
FPS)
}
if (teleportCooldown > 0) teleportCooldown--;
```

Così ogni 5 secondi al massimo si può teleportare. Questo è facile ma dipende dal framerate per la durata; usare DateTime è più robusto.

# Best practices di sviluppo mod SHVDN

Sviluppare script complessi richiede attenzione a **stabilità, manutenibilità e performance**. Ecco alcuni consigli:

- **Prevenire crash e memory leak:** Molti crash derivano da oggetti non caricati o non gestiti. Assicurarsi di:
    - **Caricare modelli e asset prima dell'uso**: usare `.Request()` sui `Model` e attendere `IsLoaded` (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) prima di spawnare ped/veicoli, altrimenti un modello non disponibile può crashare il gioco.
    - **Rilasciare risorse non più necessarie**: chiamare `MarkAsNoLongerNeeded()` su modelli, entità o asset particellari una volta che non servono più (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Questo aiuta il gioco a liberare memoria grafica.
    - **Pulire entità create**: se generate molti oggetti (ped, veicoli, props), conservatene i riferimenti in liste e rimuoveteli quando non servono più: ad es. `createdPed.Delete()` quando un NPC temporaneo muore o il potere termina. Starman mostra un approccio: tiene liste `createdPeds` e `createdProps` e li *marka* come non necessari dopo averli creati (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (anche se poi ci si potrebbe spingere a eliminarli del tutto).
    - **Evitare loop intensivi in Tick**: il metodo `OnTick` gira ogni frame (~60 volte al secondo). Operazioni pesanti (es. scansione di tutti i ped della mappa, calcoli eccessivi) vanno limitate. Si può eseguire certe logiche ogni N tick (ad es. controllare i ped vicini ogni 10 tick invece che ogni frame).
    - **Uso di Script.Wait**: nelle funzioni che eseguono compiti sequenziali (es. animazioni, pause) usare `Script.Wait(ms)` invece di bloccare in un ciclo vuoto, per non congelare il gioco (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ) (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ). Ad esempio, per far lampeggiare qualcosa ogni secondo, meglio un piccolo script separato col suo loop che usa Wait.
    - **Log degli errori**: SHVDN genera un file ScriptHookVDotNet.log con errori di script. Quando qualcosa "non funziona", controllare lì è fondamentale (GTA X

Scripting - JulioNIB mods). Per prevenire crash, si possono mettere try-catch attorno a parti critiche e almeno fare `UI.Notify("Errore: "+ex.Message)` per sapere che è successo invece di far crashare tutto.

- **Design modulare e manutenibile:** Con tanti poteri, evitare spaghetti code è fondamentale. Un pattern utile è creare una **classe base Power** con metodi virtuali `Activate()`, `Deactivate()`, `OnTick()` ecc., e poi derivare le classi specifiche (FlightPower, SuperStrengthPower, ecc.) da essa. In questo modo:
  - Ogni potere gestisce internamente il proprio stato (attivo/non attivo, eventuale durata, cooldown).
  - Lo script principale può mantenere una lista o dizionario di poteri disponibili per il player e delegare l'aggiornamento. Ad esempio:

    ```
    List<PowerBase> powers = new List<PowerBase>() { new FlightPower(),
    new StrengthPower() ... };
    ...
    // Nel Tick generale
    foreach(var p in powers) {
        if(p.IsActive) p.OnTick();
    }
    ```

    E quando un tasto viene premuto per attivare un potere, si chiama `p.Activate()`, magari disattivando altri incompatibili.
  - Si facilita così l'**estensione**: aggiungere un nuovo potere significa creare una nuova classe che implementa quelle funzioni, senza toccare troppo il resto del codice.
  - Un esempio concettuale può essere intravisto nella struttura del mod Flash di JulioNIB, che aveva un **power wheel** per selezionare i poteri (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle) (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle): internamente, ogni potere (tornado, lancio di fulmini, fase attraverso oggetti, *heart attack* per togliere il cuore ai ped) era programmato separatamente e attivabile. Probabilmente Julio ha implementato questi poteri come funzioni modulari all'interno dello script, attivabili tramite l'input dalla ruota (che settava uno stato tipo `currentPower = TornadoHands`, ecc.). Seguire un approccio OOP con classi rende ciò più pulito.
  - **Gestione dello stato**: per evitare conflitti, definire chiaramente quali poteri si escludono a vicenda. Ad esempio, volo e supervelocità a terra non possono essere attivi insieme (se voli non corri). Si possono impostare priorità o semplicemente spegnere uno quando l'altro si accende.
- **Efficient Debugging & Logging:** Durante lo sviluppo, usare **UI.Notify** per avere feedback in tempo reale (es. "Modo volo attivato", o stampare valori variabili) ( Hex for developers [.NET] - GTA5-Mods.com ). Per debug più approfondito, SHVDN supporta la console (se abilitata, premendo tilde aperta la window console). Inoltre, considerare di scrivere su file di log personalizzati: usando le classi .NET standard (System.IO) si può appendere testo a un file come debug. Togliere o disabilitare questi log nella release finale per non impattare performance. Un'altra tecnica è usare colori o effetti sul player per stati di debug (es. rendere il ped semitrasparente se una condizione è true, ecc.) – creativa ma visibile immediatamente.

- **Test incrementale:** Quando si aggiungono funzionalità, testarle una alla volta. Ad esempio, se implementate il potere di volo, provate quello isolatamente prima di aggiungere anche il tiro di ragnatela. In caso di crash o malfunzionamenti, sarà più facile capire quale modulo causa il problema. La **ricarica veloce dei script** (premendo *Insert* in gioco) aiuta molto: consente di ricompilare il dll e ricaricarlo senza riavviare GTA (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ) (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ). Impostate Visual Studio con post-build event che copia il .dll direttamente nella cartella *scripts*, così con Alt+Tab e Insert testate subito le modifiche (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ).
- **Compatibilità e aggiornamenti:** Tenete d'occhio gli aggiornamenti di SHVDN e GTA V. Dopo le patch del gioco, assicurarsi di usare la versione aggiornata di ScriptHookV e ScriptHookVDotNet, altrimenti i mod possono smettere di funzionare o crashare. JulioNIB raccomanda di avere sempre il gioco aggiornato perché i suoi mod usano animazioni/FX delle ultime patch (GTA X Scripting - JulioNIB mods). Inoltre, quando GTA V si aggiorna, è possibile che SHVDN3 impieghi qualche giorno a ricevere un update compatibile – pianificate di compilare i vostri mod con le nuove librerie se necessario.

# Raccolta di risorse utili

Per approfondire, ecco **link a guide, repository e mod esistenti** relativi ai superpoteri in GTA V, utili sia come riferimento tecnico che per trarre ispirazione:

- **Documentazione e guide ufficiali:**
  - *GitHub Wiki di ScriptHookV .NET*: contiene guide su *primi passi*, esempi di codice (cambio modello, teletrasporto) e suggerimenti su come chiamare nativi (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Ottimo per chi inizia con SHVDN.
  - *Community Script Hook V .NET – Classi e metodi*: documentazione auto-generata dell'API SHVDN v3. Ad esempio, la pagina della classe `Ped` elenca tutte le proprietà/metodi disponibili (salute, armi, task, ecc.) (Class Ped | Community Script Hook V .NET ) (Class Ped | Community Script Hook V .NET ), e quella di `World` mostra metodi come `World.AddExplosion` e `World.AddRope` (ScriptHookVDotNet v3 | Community Script Hook V .NET).
  - *Alloc8or GTA V Native DB*: database completo delle funzioni native di GTA V, con nomi, hash e parametri (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub). Indispensabile quando `Function.Call` è necessario (es: cercando *SET_ENTITY_VISIBLE* troviamo come usarlo per invisibilità).
  - *Forum GTAForums – sezione Coding*: ci sono discussioni storiche su ScriptHookVDotNet, soluzioni a problemi comuni e snippet. Ad esempio, thread su come aumentare la velocità del ped (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle) o usare `World.GetNearbyPeds` in VB/C# (Function args to PED::GET_PED_NEARBY_PEDS? - GTAForums). Anche se alcuni post sono datati, molte dritte valgono ancora.

- **Codice di mod esistenti (open source):**
    - *Starman invincibility mod (theandrew61/Starman su GitHub)* – Mod open-source che dà invincibilità temporanea stile Mario stella (GitHub - theandrew61/Starman: A mod for GTA V that grants you invincibility like in Mario games.) (GitHub - theandrew61/Starman: A mod for GTA V that grants you invincibility like in Mario games.). Utile per vedere un esempio completo di script SHVDN3 in C#, con gestione di timer, effetti (particelle scintille) (Starman/Main.cs at master · theandrew61/Starman · GitHub), musica (usa NAudio), barra HUD (Starman/Main.cs at master · theandrew61/Starman · GitHub) e disattivazione ragdoll (Starman/Main.cs at master · theandrew61/Starman · GitHub). Link: (GitHub - theandrew61/Starman: A mod for GTA V that grants you invincibility like in Mario games.).
    - *GTA V Superman Ultimate (Nexusmods)* – Mod di Superman che include volo, super-forza, laser, gelo, ecc. È script .NET; su Nexus c'è la pagina (Superman Ultimate Script mod at Grand Theft Auto 5 Nexus) con descrizione e talvolta file .ini configurazione. Alcune versioni precedenti erano open-source o hanno code snippet pubblicati su forum.
    - *JulioNIB's scripts (closed source ma documentati)* – JulioNIB è autore dei più famosi mod (Iron Man, Hulk, Thanos, Flash, Spider-Man). Sul suo blog gtaxscripting.blogspot.com pubblicava **changelog e feature list** dei mod, che aiutano a capire cosa implementare. Ad esempio, la pagina del mod *Superman* elenca poteri realizzati (corsa veloce, resistenza, volo, laser, rigenerazione) e quelli in sviluppo (soffio gelido, sollevare veicoli, visione a raggi X) (GTA X Scripting - JulioNIB mods: GTA V Superman script mod) (GTA X Scripting - JulioNIB mods: GTA V Superman script mod). La pagina del *Flash mod v2* elenca tutti gli attacchi e migliorie introdotte (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle). Anche se il codice non è visibile, sapere *cosa fa* un mod può guidare il nostro sviluppo. Alcuni link utili:
        - JulioNIB Mods Setup Guide e FAQ (Just Cause 3 script mod - Grappling hook for GTA 5) (GTA X Scripting - JulioNIB mods) (risolve problemi comuni di caricamento script, conflitti, ecc.).
        - **Grappling hook mod** (Just Cause style) – utilissimo per implementare ragnatele: descrizione delle funzionalità di rampino (aggancio a muri, spinta col paracadute, attacco a veicoli/peds) (GTA X Scripting - JulioNIB mods: GTA V - Just Cause 2 Grappling hook mod) (GTA X Scripting - JulioNIB mods: GTA V - Just Cause 2 Grappling hook mod).
        - **Green Goblin mod** – combina gadget (bombe, mitragliatrice sul deltaplano) e volo su veicolo speciale. Vedi descrizione su GTA5-Mods ( Green Goblin Glider and Powers Script - GTA5-Mods.com ).
        - **Iron Man script mod v2** – presentato su PCGamer con dettagli su nuove animazioni ed effetti (GTA 5 Iron Man script mod launches awesome Version 2.0 | PC Gamer) (GTA 5 Iron Man script mod launches awesome Version 2.0 | PC Gamer). Mostra ad esempio l'HUD stile casco di Iron Man e mosse come il ground slam.

        - **Thanos (Endgame) mod** – Julio ha fatto anche Thanos con le gemme: anche qui, leggere le feature (portali, telecinesi, "snap" che dissolve ped) può dare spunti su come fare teleport o dissolvenze (es. trasformare ped in cenere è ottenibile applicando `Ped.IsInvincible = false` e una routine che riduce la salute lentamente mentre spawn particelle di polvere).
    - *Psychokinetic mod (WIP by sollaholla)* – mod .NET in sviluppo che implementa **telecinesi generica** e tante varianti di poteri (ghiaccio, fuoco, elettricità) ( Psychokinetic [W.I.P] - GTA5-Mods.com ) ( Psychokinetic [W.I.P] - GTA5-Mods.com ). Non è open-source, ma nel readme (visibile su GTA5-Mods) c'è spiegato il funzionamento: ad es. *GrabPed* (F4), *Push/Pull* (Shift+X / Ctrl+X) per scagliare entità, *LockOn* (Shift+E) per fissare un bersaglio prima di sollevarlo ( Psychokinetic [W.I.P] - GTA5-Mods.com ) ( Psychokinetic [W.I.P] - GTA5-Mods.com ). Interessante la parte di **Flying** con 3 modalità ( Psychokinetic [W.I.P] - GTA5-Mods.com ) e *Transform* (Shift+T) – possiede potenzialmente una trasformazione del player. Inoltre, cita un "teleport almost anywhere you look" con middle mouse ( Psychokinetic [W.I.P] - GTA5-Mods.com ). Questo mod dimostra quante cose si possono fare combinando i controlli. Nel changelog avanzato ( Psychokinetic [W.I.P] - GTA5-Mods.com ) ( Psychokinetic [W.I.P] - GTA5-Mods.com ) parla di animazioni di *crush* (schianto) personalizzate e di come hanno implementato il controllo di armi rimosse ai ped (telecinesi anche sulle armi!). **Sebbene il codice non sia pubblico, questo readme è quasi un design document pieno di idee.**
- **Community e tutorial video:**
    - *YouTube – ScriptHookV.NET scripting series* – esistono video tutorial in italiano e inglese. In italiano trovi soprattutto guide di installazione, ma per lo scripting C# le risorse sono quasi tutte in inglese. Un'ottima serie è "**GTA V ScriptHookV Coding**" (canale *Tutorials();*): parte 1 impostazione ambiente (ScriptHookV Tutorials - YouTube), parte 2 primo mod (spawn veicolo e ripararlo) (GTA:V ScriptHookV Coding: Part Two | Your First Mod! - YouTube), parte 3 teletrasporto (GTA:V ScriptHookV Coding: Part Three | Teleport Player ... - YouTube), parte 4 etc. Segue un approccio didattico dal base all'avanzato. Un altro youtuber, *Games & Graphics*, fa tutorial su installazione mod e talvolta mostra come usare NativeUI e SHVDN (per es. ha uno script di esempio su come creare un menu trainer).
    - *Forum GTA5-Mods* – Sezione "Tutorials" contiene qualche guida (es. **Hex coding for developers** dove un utente mostra come leggere coordinate con UI.Notify ( Hex for developers [.NET] - GTA5-Mods.com )). Anche la sezione "Script & Plugin Development" ha thread di domanda/risposta – ad esempio, un thread su come aumentare la velocità di corsa con esempi di codice (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle).
    - *Reddit r/GTAV_mods* – utile per aggiornamenti (ad es. quando escono nuove versioni di ScriptHookV e .NET, pin delle nightly builds (Scripthookv not working properly : r/GTAV_Mods - Reddit)) e per vedere cosa chiede la gente. Meno tecnico di GTAForums, ma a volte emergono fix (come nel caso di crash della mod Spider-Man risolto usando una versione meno recente di SHVDN (Spiderman Power Crashes

game, on use. : r/GTAV_Mods - Reddit) – segno che la compatibilità delle versioni conta).

- **Strumenti e risorse aggiuntive:**
  - *NativeUI library*: per creare menu, liste, e HUD elementari facilmente. Molti mod (.dll) richiedono NativeUI.dll. Se vuoi fare un menu per selezionare personaggi o poteri, NativeUI offre una struttura pronta di menu navigabili e personalizzabili.
  - *OpenIV & addon peds*: se il tuo mod prevede l'uso di modelli personalizzati (es. modelli 3D di Iron Man, Hulk), devi fornire istruzioni per installarli (spesso tramite **AddonPeds** mod). La guida di JulioNIB su come aggiungere ped (GTA X Scripting - JulioNIB mods) o il tool *AddonPeds* su gta5-mods ti saranno utili per testare con i model corretti.
  - *ScriptHookVDotNet source code*: Dare un'occhiata al repository SHVDN (scripthookvdotnet/source/scripting_v2/GTA/Entities/Entity.cs at main) può chiarire come funzionano certe funzioni. Ad esempio, si può vedere come è implementato `Ped.ApplyForce` dentro SHVDN (scripthookvdotnet/source/scripting_v2/GTA/Entities/Entity.cs at main) (in realtà chiama il nativo `APPLY_FORCE_TO_ENTITY`).

In conclusione, creare la mod **MarvelScript** con tutti questi poteri richiederà tempo e sperimentazione. **Studiare i mod esistenti**, provare snippet in-game e seguire le best practice sopra ti aiuterà a evitare problemi. Con ScriptHookVDotNet3 hai un'ottima base per realizzare poteri spettacolari (volo, velocità, colpi, teletrasporti) in modo stabile e compatibile con gli script per **Player e NPC**. Buon coding e divertiti a trasformare Los Santos nel tuo sandbox di supereroi!

**Riferimenti:** Guide ufficiali e codice di esempio (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (How Tos · scripthookvdotnet/scripthookvdotnet Wiki · GitHub) (Getting started with ScriptHookVDotNet | Community Script Hook V .NET ), note dai mod di JulioNIB (Superman, Flash) (GTA X Scripting - JulioNIB mods: GTA V Superman script mod) (GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle), progetti open-source come *Starman* (Starman/Main.cs at master · theandrew61/Starman · GitHub) (Starman/Main.cs at master · theandrew61/Starman · GitHub) e documentazione ScriptHookVDotNet v3 (Class Ped | Community Script Hook V .NET ) (Class World | Community Script Hook V .NET).