

Perfetto! Procederò con un'espansione approfondita del documento, aggiungendo esempi pratici, ulteriori poteri sovrumanici, spiegazioni dettagliate passo-passo e una progressione che parte dai concetti base fino a tecniche avanzate. Il materiale sarà pensato per principianti, ma arriverà a coprire anche argomenti di livello professionale, con una struttura chiara e didattica.

Appena pronto, ti farò avere la nuova versione estesa del documento qui in chat.

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

Grand Theft Auto V non prevede “superpoteri” di base, ma grazie alle mod script è possibile trasformare il gioco in un vero sandbox di supereroi. In questa guida completa vedremo come creare una *mod* in C# usando **ScriptHookV** e **ScriptHookVDotNet3** per implementare abilità sovrumaniche (volo, super velocità, forza sovrmana, teletrasporto, ecc.), partendo dai concetti più semplici fino a tecniche avanzate. La guida è strutturata come un tutorial passo-passo: inizieremo con la configurazione dell’ambiente e la struttura base di uno script, per poi aggiungere gradualmente nuove funzionalità e poteri, spiegando in dettaglio ogni concetto incontrato. Verranno forniti esempi di codice commentati per ogni abilità, oltre a consigli sulle *best practice* (organizzazione del codice, gestione di HUD e compatibilità) e suggerimenti per il debugging con strumenti esterni (Visual Studio, **OpenIV**, **NativeUI**, ecc.). Infine, faremo riferimenti a mod famose di supereroi in GTA V, spiegando come implementano certe funzionalità, per trarre ispirazione nella creazione di una mod completa (ad esempio un progetto “MarvelScript” che racchiuda più poteri).

Nota: le mod script funzionano solo in **GTA V modalità single-player** (mai online). Assicurarsi di avere l’ultima versione di ScriptHookV e ScriptHookVDotNet (v3) compatibili con la propria versione di gioco. Inoltre, molte funzionalità richiedono modelli o effetti personalizzati: useremo alcuni modelli base di GTA V, ma per poteri avanzati (es. trasformarsi in Hulk o Iron Man) dovrete installare i relativi *addon* di modelli usando OpenIV. L’obiettivo però è focalizzarsi sulla programmazione dei poteri.

Prerequisiti e configurazione dell’ambiente

Prima di iniziare a scrivere codice, è necessario configurare l’ambiente di modding per GTA V:

- **ScriptHookV**: la libreria di Alexander Blade necessaria per caricare *script native* in GTA V. Scaricatela dal sito ufficiale e copiate **ScriptHookV.dll** e l’ASI loader (**dinput8.dll**) nella cartella di gioco principale ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)).
- **ScriptHookVDotNet3**: il *runtime* che permette di eseguire script .NET (C# o VB) in GTA V. Scaricate l’ultima release (o nightly) dal repository GitHub e copiate i file **ScriptHookVDotNet.asi** e **ScriptHookVDotNet3.dll** nella cartella di gioco ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)). Create inoltre la cartella **scripts** nella directory di GTA V, se non esiste già, dove inseriremo i nostri script .dll o .cs ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)).

- **SDK / Documentazione:** Per sviluppare serve il file **ScriptHookVDotNet3.dll** come riferimento. Potete creare un progetto C# in Visual Studio e aggiungere un riferimento a questo .dll per avere intellisense e definizioni delle classi. Assicuratevi di compilare con .NET Framework compatibile (SHVDN3 usa .NET Framework 4.8). In alternativa, potete scrivere direttamente file .cs e metterli in **scripts** (il runtime li compilerà automaticamente), ma usare Visual Studio facilita il debugging.

Una volta configurati questi elementi, possiamo passare alla creazione del nostro primo script. *Tip:* Durante lo sviluppo, è possibile ricaricare rapidamente gli script senza riavviare il gioco: basta ricompilare il progetto e copiare la DLL aggiornata nella cartella **scripts**, quindi premere il tasto **Insert** in gioco per ricaricare gli script ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)). Questo accelera enormemente i test iterativi.

Struttura base di uno script .NET per GTA V

Ogni script .NET per GTA V deve derivare dalla classe base **Script** fornita da **ScriptHookVDotNet** ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)). Questa classe offre tre eventi fondamentali su cui lavorare ([Getting started with ScriptHookVDotNet | Community Script Hook V .NET](#)):

- **Tick** – Evento che viene chiamato ad ogni frame di gioco (o ad intervalli regolari configurati) e che serve per la logica continua, ad esempio aggiornare lo stato di un potere attivo, applicare effetti nel tempo, controllare condizioni continuamente.
- **KeyDown** – Evento chiamato quando un tasto viene premuto.
- **KeyUp** – Evento chiamato quando un tasto viene rilasciato.

Tipicamente, uno script definisce nel costruttore la sottoscrizione a questi eventi, associando dei metodi handler. Ecco un esempio minimale di script che mostra la struttura e come intercettare l'input da tastiera ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)):

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

public class CreateVehicle : Script {
    public CreateVehicle() {
        // Registriamo gli handler per gli eventi Tick e input da tastiera
        Tick += OnTick;
        KeyUp += OnKeyUp;
    }

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

    private void OnKeyUp(object sender, KeyEventArgs e) {
        if (e.KeyCode == Keys.NumPad1) {
            // Esempio: spawn di un veicolo davanti al player
            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 la posizione e direzione forward ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)). Premendo **NumPad1** spawniamo un’auto di tipo *Zentorno* a qualche metro di distanza di fronte al giocatore, e mostriamo un messaggio su schermo con **UI.Notify**. Questo semplice script ci insegna alcuni concetti chiave:

- Come derivare la classe **Script** e usare i **metodi evento** **OnTick**, **OnKeyUp/OnKeyDown**.
- Come accedere al **mondo di gioco**: la classe statica **World** fornisce metodi per creare entità di gioco, come **World.CreateVehicle** usato sopra per spawnare un veicolo.
- Come accedere al **giocatore**: tramite **Game.Player.Character** otteniamo il *Ped* del giocatore controllato.
- Utilizzo di funzioni di **UI**: la classe **UI** offre metodi come **Notify** per mostrare messaggi (utile per debugging) e anche funzioni per disegnare testo o forme sullo schermo.

Notate che non c’è un metodo **Main**: l’entry point è il costruttore dello script che viene chiamato dal runtime di SHVDN quando il gioco carica lo script (all’avvio o alla pressione di Insert per ricaricare). Il nostro script rimane in esecuzione finché il gioco è aperto o finché viene disabilitato (ad esempio se usiamo la console per *abortire* uno script).

Gestione degli eventi di gioco e input utente

Nell’esempio sopra abbiamo visto come catturare un input da tastiera tramite **KeyUp**. La gestione dell’**input** è cruciale per attivare i poteri: dovremo scegliere quali tasti attivano o disattivano una certa abilità, e intercettarli nello script. Alcuni consigli sulla gestione input:

- **Distinguere KeyDown/KeyUp**: Usare **KeyDown** se vogliamo rilevare il *tenere premuto* un tasto (ripetuto ogni frame finché è premuto), oppure **KeyUp** se vogliamo agire solo al singolo impulso (quando l’utente preme e rilascia). Ad esempio, per un potere “modalità attiva/disattiva” conviene usare **KeyUp** (un click attiva, un altro disattiva), mentre per un potere che deve durare solo finché il tasto è tenuto premuto (es. un raggio laser continuo) potrebbe essere più adatto **KeyDown**.
- **Game.IsKeyPressed**: In alternativa agli eventi, SHVDN offre metodi per controllare lo stato dei tasti in qualunque momento. Ad esempio **Game.IsKeyPressed(Keys.X)** può essere usato dentro **Tick** per verificare se un certo tasto è premuto e reagire di conseguenza ([Implementare poteri sovraumani GTA V.pdf](#)). Nell’esempio sul volo controllato più avanti useremo questo approccio per leggere input di movimento in tempo reale.
- **Input da controller**: Gli eventi **KeyUp/Down** funzionano solo per tastiera. Se volete supportare i controller, potete usare i metodi di Gamepad mapping di GTA (ad esempio **Game.IsControlPressed(...)** con i codici di controlli). Documentarsi sui **Control ID** di GTA V se necessario. Per questa guida ci concentreremo su input da tastiera per semplicità.
- **Evita conflitti**: Scegli tasti che non interferiscono con altri controlli di gioco o altre mod. Per sicurezza, consenti all’utente di personalizzare i tasti tramite un file **.ini**

(ScriptHookVDotNet offre la classe `ScriptSettings` per leggere configurazioni facilmente). Ad esempio, potremmo leggere da un file di config il tasto da usare per attivare ogni potere, rendendo la mod più flessibile.

Un'ultima nota: è possibile impostare la proprietà `Interval` dello script (ereditata da `Script`) per regolare l'intervallo di chiamata dell'evento Tick. Di default Tick viene chiamato ogni frame; se poniamo `Interval = 10` nel costruttore, verrà chiamato ogni 10 millisecondi circa. Questo può essere utile per modulare la frequenza di aggiornamento di alcuni effetti (ad esempio potremmo non aver bisogno di controllare certe cose ogni singolo frame). In generale si può lasciare il default e gestire manualmente le cadenze nel codice (ad es. usando un contatore di frame).

Chiamare funzioni native di GTA V

ScriptHookVDotNet espone molte funzionalità di gioco attraverso le sue classi e metodi (come `World.CreateVehicle` visto sopra). Tuttavia, per alcune operazioni avanzate potrebbe essere necessario chiamare direttamente i **nativi** del motore di GTA V, cioè quelle funzioni proprie del gioco usate nello scripting originale. SHVDN permette di farlo tramite la classe `GTA.Native.Function` e in particolare il metodo generico `Call<Hash>` ([Implementare poteri sovraumani GTA V.pdf](#)). In pratica, ogni funzione nativa di GTA ha un *hash* identificativo; ScriptHookV fornisce questi hash (tramite l'enum `Hash` in SHVDN) e consente di invocare la funzione corrispondente con i parametri richiesti.

Esempio: non esiste un metodo managed pronto per disattivare la radio di un veicolo, ma sappiamo che il motore GTA ha la funzione nativa `SET_VEHICLE_RADIO_ENABLED(vehicle, false)`. Possiamo chiamarla così in C#:

```
Vehicle car = ... // riferimento al veicolo
Function.Call(Hash.SET_VEHICLE_RADIO_ENABLED, car, false);
```

Allo stesso modo, moltissimi effetti speciali o comportamenti (es. applicare forze complesse, generare particelle, manipolare il tempo di gioco) si ottengono chiamando nativi specifici. Per trovare i nativi e i relativi parametri, si può consultare l'**Native DB** aggiornata (ad esempio la community database di alloc8or) o la wiki di ScriptHookVDotNet che mostra spesso come certe funzioni .NET sono implementate proprio chiamando i nativi sottostanti ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)). Più avanti useremo `Function.Call` per poteri avanzati (come modificare il timescale del gioco per rallentare il tempo, o creare effetti esplosivi personalizzati).

Gestione della memoria: quando si chiamano nativi o si creano oggetti (veicoli, ped, effetti) è importante considerare la gestione delle risorse. GTA V ha un limite di entità attive; se il nostro script genera decine di oggetti ogni secondo senza mai eliminarli, potremmo saturare la memoria del gioco o causare instabilità. SHVDN fornisce metodi per pulire gli oggetti non più usati, ad esempio `Vehicle.Delete()` per rimuovere un veicolo, oppure `Model.MarkAsNoLongerNeeded()` per liberare un modello caricato in memoria ([Implementare poteri sovraumani GTA V.pdf](#)). È buona abitudine, quando un potere termina, ripristinare o rimuovere gli oggetti creati: ad esempio, se il nostro potere “scudo energetico” spawnna un oggetto scudo, all’atto di disattivarlo dovremmo cancellarlo. Fortunatamente il garbage collector di .NET libera la memoria gestita automaticamente, ma le entità di gioco vanno gestite esplicitamente.

Performance: un frame di GTA V dura circa 16ms (se il gioco gira a ~60fps). Il codice nel Tick deve essere efficiente per non causare cali di frame-rate. Alcuni consigli di ottimizzazione:

- Evitare operazioni troppo pesanti in Tick. Se un calcolo può essere fatto solo al momento dell'attivazione del potere (es. caricamento di un modello, calcolo di coordinate iniziali), fatelo una tantum, non ad ogni frame.
- Limitare i loop su grandi collezioni in Tick. Ad esempio, iterare su *tutti* i ped nel mondo ogni frame è costoso. Meglio ridurre la portata (es. solo ped entro un raggio dall'eroe) o la frequenza (es. controllare ogni 10 tick invece che ogni tick).
- Usare bool di stato per evitare di eseguire codice quando non necessario. Ad esempio, se il potere di volo è disattivato, nel Tick potete direttamente saltare la logica di volo per quel frame, così da non sprecare cicli.
- Alcuni effetti possono essere eseguiti a frequenza ridotta: ScriptHookVDotNet consente di usare `Script.Wait(ms)` per ritardare l'esecuzione di un blocco senza congelare il gioco. In uno script potete anche lanciare routine separate (coroutine) che eseguono codice e poi fanno Wait. Per semplicità qui non approfondiremo, ma sappiate che è possibile avere più loop che girano (ad esempio uno script potrebbe avviare un secondo loop che gestisce un effetto temporizzato in parallelo) ([Implementare poteri sovraumani GTA V.pdf](#)).

Con queste basi in mente, passiamo all'implementazione dei **poteri sovrumani** veri e propri. Seguiremo una progressione logica: inizieremo con abilità relativamente semplici da ottenere combinando poche proprietà/nativi, per poi introdurre poteri via via più complessi. Ogni sezione presenterà il concetto, spiegherà come GTA V può supportarlo, mostrerà un esempio di codice e fornirà consigli specifici.

Implementazione di poteri sovrumani: tutorial passo-passo

In questa sezione realizzeremo una serie di poteri stile supereroe. Per ognuno, immaginiamo di aggiungerlo al nostro eroe in GTA V, costruendo progressivamente una mod completa. Potete scegliere di implementare i poteri come script separati oppure (metodo consigliato) integrarli in un unico progetto modulare – di cui parleremo nelle best practice. Iniziamo!

Super salto (balzo potenziato)

Il “super salto” è un potere semplice ma divertente: consente al giocatore di spiccare balzi molto più alti del normale, come un supereroe dalla forza sovrumana nelle gambe. In GTA V il salto del player è normalmente limitato dalla fisica e animazione; non c’è un parametro facile da impostare per aumentarlo, ma possiamo ottenere l’effetto applicando una **forza verso l’alto** al ped al momento del salto.

Approccio: rilevare quando il giocatore prova a saltare e, in quel momento, impartire una spinta aggiuntiva verso l’alto al personaggio. GTA V ha una proprietà `Ped.IsJumping` che indica se un ped (incluso il player) è in fase di salto. Possiamo controllarla in `Tick` e, se diventa true, applicare la forza.

Ad esempio, potremmo assegnare il super salto al tasto **Spazio** (lo stesso del salto normale). Quando il giocatore preme Spazio, GTA fa saltare il ped – il nostro script intercerterà l’evento e amplificherà il salto. In pseudocodice:

```

if (Game.IsKeyPressed(Keys.Space) && !playerPed.IsJumping && playerPed.IsOnFoot)
{
    // il player ha premuto salto ed è a piedi (non in veicolo)
    playerPed.Task.Jump(); // forziamo l'animazione di salto (opzionale, GTA lo
fa già)
}
if (playerPed.IsJumping && !gaveExtraBoost) {
    // una tantum quando inizia il salto, applichiamo spinta extra
    playerPed.ApplyForce(new Vector3(0, 0, 5.0f));
    gaveExtraBoost = true;
}
// Quando atterra (non è più Jumping), resettiamo il flag
if (!playerPed.IsJumping) {
    gaveExtraBoost = false;
}

```

Nella pratica potremmo fare l'`ApplyForce` solo la prima volta per salto (controllato da un flag booleano `gaveExtraBoost`). La forza di 5.0f è da tarare: aumentandola il ped salterà più in alto. `ApplyForce` è un metodo di SHVDN (ereditato da `Entity`) che sotto al cofano chiama il nativo `APPLY_FORCE_TO_ENTITY` ([Implementare poteri sovraumani GTA V.pdf](#)). Nel nostro caso applichiamo una forza verso l'alto (asse Z).

Un punto importante: disabilitare temporaneamente la gravità del ped può aiutare a farlo salire più in alto e ricadere più dolcemente. Ad esempio, potremmo fare `playerPed.HasGravity = false` appena dopo aver applicato la forza, e riabilitare la gravità quando il ped inizia a ridiscendere o dopo un breve timer. Tuttavia, attenzione a riattivare sempre la gravità, altrimenti il ped potrebbe restare a fluttuare!

Esempio di codice completo per super salto:

```

bool gaveExtraBoost = false;

public void OnTick(object sender, EventArgs e) {
    Ped player = Game.Player.Character;
    // Se premiamo Spazio mentre siamo a terra, forziamo il salto
    if (Game.IsKeyPressed(Keys.Space) && !player.IsJumping && player.IsOnFoot) {
        player.Task.Jump();
    }
    // All'inizio del salto, una sola volta diamo spinta extra
    if (player.IsJumping && !gaveExtraBoost) {
        player.ApplyForce(new Vector3(0, 0, 6.0f)); // spinta verso l'alto
        player.HasGravity = false; // temporaneamente niente
gravità
        gaveExtraBoost = true;
    }
    // Se il ped sta discendendo o ha finito il salto, ripristina gravità
    if (gaveExtraBoost && !player.IsJumping) {
        player.HasGravity = true;
        gaveExtraBoost = false;
    }
}

```

Con questo script, il nostro personaggio farà salti molto più alti del normale. Si può migliorare aggiungendo effetti visivi all'atterraggio (magari una piccola scossa a terra) ma terremo questo spunto per la sezione forza sovrumana/colpi a terra. Intanto, passiamo a dare velocità incredibile al nostro eroe.

Super velocità (correre come Flash)

Chi non vorrebbe correre attraverso Los Santos alla velocità del fulmine? Per implementare la **super velocità** del giocatore, possiamo agire su due fronti: aumentare la velocità di movimento base e aggiungere effetti scenici (scia, distorsione) per dare la sensazione di rapidità estrema.

1. Aumentare la velocità di corsa: GTA V permette di modificare il moltiplicatore di velocità per il *player* attraverso un nativo. Il nativo

`SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER(player, multiplier)` consente di aumentare la velocità di corsa e scatto del giocatore ([Implementare poteri sovraumani GTA V.pdf](#)). Normalmente il multiplier è 1.0; se lo portiamo, ad esempio, a 1.5, il player correrà ~50% più veloce. Possiamo chiamarlo con SHVDN:

```
Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER, Game.Player, 1.5f);
```

Questo effetto è globale e permane finché non viene rimesso a 1.0, quindi il nostro potere “super velocità” dovrà attivarlo e disattivarlo appropriatamente. Inoltre, c’è anche

`SET_SWIM_MULTIPLIER_FOR_PLAYER` se volessimo super velocità a nuoto.

2. Effetto “Flash time” (slow-motion relativo): Nei fumetti/film, quando Flash corre tutto intorno a lui sembra rallentare. Per simulare questa percezione possiamo usare il nativo

`SET_TIME_SCALE` che modifica la velocità del tempo di gioco ([Implementare poteri sovraumani GTA V.pdf](#)). Ad esempio, impostare `SET_TIME_SCALE(0.5)` fa andare il gioco a metà velocità (tutto rallentato). Se mentre il mondo è al 50% noi aumentiamo la velocità del player, daremo l’impressione che l’eroe si muova a velocità normale mentre tutto il resto è al rallentatore – in pratica, dal suo punto di vista sta andando super veloce. Possiamo quindi, all’attivazione di super velocità, abbassare leggermente il timescale (es. 0.5) per un effetto *bullet-time*. **Nota:** time scale influisce su *tutto* (fisica, audio, ecc.). Valori estremi (0.1 o 0.0) possono causare comportamenti strani e non permettere al player di muoversi correttamente. Consigliamo di non scendere sotto 0.3 per mantenere controlli reattivi.

3. Controllo e spinta in corsa: Oltre al multiplier, potremmo voler dare una spinta in avanti durante la corsa, per superare i limiti del gioco. Un trucco è applicare una piccola forza in avanti ogni frame in cui il player sta correndo. Ad esempio, se `playerPed.IsRunning` è true, potremmo fare `playerPed.ApplyForce(playerPed.ForwardVector * 5.0f)` ([Implementare poteri sovraumani GTA V.pdf](#)). Ciò spinge il ped in avanti, aumentandone ulteriormente la velocità oltre il limite delle animazioni. Bisogna fare attenzione a non esagerare o il giocatore potrebbe perdere il controllo in curva.

4. Effetti grafici (facoltativi): Per enfatizzare la velocità, possiamo generare una *scia* dietro al player. Un modo semplice è utilizzare un’esplosione “fumogena” invisibile che lascia una nube di polvere. Nel nostro esempio useremo `World.AddExplosion` con tipo Smoke e forza nulla, così da avere solo fumo ([Implementare poteri sovraumani GTA V.pdf](#)). In alternativa, per un vero effetto scia continuo, bisognerebbe usare i **Particle FX** di GTA: ad esempio c’è l’effetto polvere sollevata usato per elicotteri o per veicoli veloci. Questi però richiedono di caricare e attivare i particle assets (ne parleremo per i laser). Per ora, una piccola nube di fumo ad ogni passo può andare.

Codice di esempio per super velocità:

Supponiamo di attivare/disattivare la super velocità premendo **Shift**. Quando attivo, aumentiamo il multiplier e abilitiamo slow-motion; quando disattivo, riportiamo tutto alla normalità.

```
bool superSpeedActive = false;

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.LShiftKey) { // tasto Shift sinistro attiva/disattiva
        superSpeedActive = !superSpeedActive;
        if(superSpeedActive) {
            // Attiva super velocità
            Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER,
Game.Player, 1.5f);
            Function.Call(Hash.SET_SWIM_MULTIPLIER_FOR_PLAYER, Game.Player,
1.5f);
            Function.Call(Hash.SET_TIME_SCALE, 0.5f); // rallenta il tempo del
gioco
            UI.Notify("Super Velocità ATTIVATA");
        } else {
            // Disattiva: ripristina valori normali
            Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER,
Game.Player, 1.0f);
            Function.Call(Hash.SET_SWIM_MULTIPLIER_FOR_PLAYER, Game.Player,
1.0f);
            Function.Call(Hash.SET_TIME_SCALE, 1.0f); // tempo normale
            UI.Notify("Super Velocità disattivata");
        }
    }
}

public void OnTick(object sender, EventArgs e) {
    if(superSpeedActive) {
        Ped player = Game.Player.Character;
        if(player.IsRunning) {
            // Spinta extra in avanti mentre corre
            player.ApplyForce(player.ForwardVector * 5.0f);
            // Effetto fumo dietro al giocatore
            Vector3 pos = player.Position - player.ForwardVector * 2; // 2 metri
            dietro
            World.AddExplosion(pos, ExplosionType.Smoke, 0.0f, 0.1f, false,
true);
        }
    }
}
```

In questo codice, quando attiviamo la modalità Flash, impostiamo i multiplier a 1.5 (si può aumentare oltre, ma già a 1.5-2.0 il controllo diventa difficile) e dimezziamo il tempo di gioco. Nel Tick, finché **superSpeedActive** è true, se il player sta correndo applichiamo la spinta e creiamo del fumo dietro di lui ([Implementare poteri sovraumani GTA V.pdf](#)). L'esplosione di tipo Smoke con intensità 0 è praticamente invisibile a parte il fumo, e con raggio 0.1 resta localizzata. Questo genera una serie di nuvolette dietro il giocatore che simulano polvere sollevata.

Da Flash a Quicksilver: Potete personalizzare ulteriormente: ad esempio, disabilitare collisioni mentre corre per attraversare gli oggetti (un effetto di *phasing*), o aumentare il campo visivo (FOV) per dare un senso di velocità. Nel mod “Flash” di JulioNIB, in versione 2.0, sono stati aggiunti moltissimi dettagli come la corsa sui muri, il *Tornado Attack* (rotazione delle braccia per creare turbine d'aria) e persino un “Heart Attack” dove Flash strappa il cuore dei nemici ([GTA 5 The Flash Version 2.0 script mod is head-spinning awesome | PC Gamer](#)). Queste mosse avanzate richiedono

animazioni custom e logiche complesse, ma il principio base – velocità aumentata e tempo rallentato – è quanto abbiamo qui implementato.

Riferimento: Il mod **Flash v2** di JulioNIB (2017) migliorò la versione originale con animazioni nuove, wall-run e attacchi speciali ([GTA 5 The Flash Version 2.0 script mod is head-spinning awesome | PC Gamer](#)). Il nostro super velocista ora è pronto; passiamo a un potere che ci porta **sopra** le strade di Los Santos: il volo.

Volo (volare come Superman)

Il volo è uno dei poteri più iconici. Implementarlo richiede di *slegare* il personaggio dalla normale gravità e controllarne la posizione manualmente. In pratica, dobbiamo fare in modo che quando il volo è attivo, il nostro personaggio non cada e possiamo spostarlo liberamente nell'aria in base all'input.

Disabilitare la gravità: La classe **Entity** (da cui **Ped** deriva) ha la proprietà **HasGravity** ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)).

Impostando **Game.Player.Character.HasGravity = false**, il ped non sarà più soggetto alla gravità e rimarrà sospeso in aria se non ha altre forze applicate. Questo è il primo passo: attivare il volo significa *spegnere* la gravità sul player (e magari renderlo invincibile ai danni da caduta, impostando **IsInvincible = true** finché vola, così se sbatte da qualche parte non muore).

Controllo del movimento in volo: Ci sono vari modi, il più semplice è usare **ApplyForce** per spingere il ped nella direzione desiderata. Possiamo leggere l'input WASD per determinare direzione di spinta:

- W (avanti) -> spinta nella direzione di visuale frontale del ped.
- S (indietro) -> spinta all'indietro (contraria alla direzione frontale).
- A/D (strafing laterale) – GTA non ha un movimento laterale per ped, potremmo ignorarli o usarli per rollio.
- Shift (su) e Ctrl (giù) per aumentare o diminuire quota.

Un esempio semplificato di controllo in volo ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)):

```
Vector3 vel = playerPed.Velocity;
if (Game.IsKeyPressed(Keys.W)) {
    playerPed.Velocity = playerPed.ForwardVector * 20f;
}
if (Game.IsKeyPressed(Keys.S)) {
    playerPed.Velocity = playerPed.ForwardVector * -10f;
}
if (Game.IsKeyPressed(Keys.Space)) {
    // verso l'alto
    playerPed.Velocity = new Vector3(vel.X, vel.Y, 10f);
}
```

In questo snippet (indicativo), premendo W portiamo la velocità a 20 in avanti, S a -10 (per frenare/retrocedere), Spazio dà velocità verso l'alto. Un codice così funziona, ma è un po' rigido; un approccio migliore è usare **ApplyForce** additivo per sommare forze e ottenere un movimento più fluido e accelerazioni progressive. Ad esempio:

`playerPed.ApplyForce(playerPed.ForwardVector * 2f)` ogni tick in cui W è premuto, così aumenta gradualmente la velocità in avanti.

Inoltre, per controllare direzione e orientamento: potremmo voler usare il **mouse** per far guardare il ped verso dove muoviamo la camera, simulando così un volo più intuitivo (come negli shooter volanti). Il mod **Superman** di JulioNIB segue questo schema: attivi il volo premendo un tasto (es. salto mentre sei in aria), poi controlli la direzione col mouse e la velocità con input avanti/dietro, volando più veloce man mano che mantieni la direzione ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)).

Prevenire animazioni indesiderate: quando togliamo gravità, il gioco a volte cerca di far animare il ped come in caduta libera o paracadute. Possiamo pulire le **tasks** correnti del ped con `playerPed.Task.ClearAll()` per interrompere animazioni precedenti ([Implementare poteri sovraumani GTA V.pdf](#)). Inoltre, potremmo forzare un'animazione di volo personalizzata (se ne abbiamo una) o riusare un'animazione esistente: ad esempio alcuni mod riutilizzano l'animazione di nuoto orizzontale in aria, che somiglia a Superman in volo ([Implementare poteri sovraumani GTA V.pdf](#)).

Evitare collisioni: durante il volo potrebbe essere frustrante collidere con ogni edificio o palo. Una scelta è rendere il ped **senza collisioni** temporaneamente (`playerPed.IsCollisionEnabled = false`) mentre vola, così può attraversare gli oggetti. Questo però potrebbe farlo entrare nelle pareti. In alternativa, mantenerlo con collisioni ma invulnerabile, così se sbatte non muore e non cade. Dipende dal risultato desiderato. JulioNIB nel suo mod Superman rende il ped invincibile e probabilmente gestisce i casi di impatto con una semplice reazione visiva ma senza danno ([Implementare poteri sovraumani GTA V.pdf](#)).

Attivazione/disattivazione del volo: possiamo mappare un tasto toggle (es. F o Invio in aria). Ad esempio, se il player preme F e non è in volo, attiviamo volo (HasGravity false, invincibilità true, salviamo uno stato); se lo ripreme, disattiviamo volo (HasGravity true). Quando si disattiva, assicurarsi che se era in aria, cominci a cadere normalmente o atterri dolcemente.

Vediamo un codice combinato per volo, assegnato ad esempio al tasto F:

```
bool flightMode = false;

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.F) {
        flightMode = !flightMode;
        Ped player = Game.Player.Character;
        if(flightMode) {
            // Attiva volo
            player.HasGravity = false;
            player.IsInvincible = true;
            player.Task.ClearAll(); // rimuove animazioni tipo paracadute
            UI.Notify("Modalità volo attivata");
        } else {
            // Disattiva volo
            player.HasGravity = true;
            player.IsInvincible = false;
            UI.Notify("Modalità volo disattivata");
        }
    }
}

public void OnTick(object sender, EventArgs e) {
```

```

if(!flightMode) return;
Ped player = Game.Player.Character;
// Controlli di volo
float speed = 1.5f; // fattore di accelerazione
if(Game.IsKeyPressed(Keys.W)) {
    player.ApplyForce(player.ForwardVector * speed);
}
if(Game.IsKeyPressed(Keys.S)) {
    player.ApplyForce(player.ForwardVector * -speed);
}
if(Game.IsKeyPressed(Keys.Space)) { // su
    player.ApplyForce(Vector3.WorldUp * speed);
}
if(Game.IsKeyPressed(Keys.ControlKey)) { // giù
    player.ApplyForce(Vector3.WorldDown * speed);
}
// (A e D per rollio o spostamento laterale potrebbero essere aggiunti)
}

```

In questa implementazione, finché `flightMode` è true, ad ogni tick applichiamo forze in base ai tasti premuti. La costante `speed` determina quanto forte acceleriamo in quella direzione – aumentando questo valore voliamo più velocemente. Si può rendere la velocità dinamica (ad es. aumentare un contatore finché W è tenuto premuto, fino a un max) per simulare accelerazione graduale.

Per mantenere il ped orientato correttamente, potremmo sincronizzare la sua rotazione con la direzione della telecamera: ad esempio, settare `player.Rotation` verso `GameplayCamera.Direction`. In SHVDN v3 si può anche usare `playerPed.LookAt(...)` o manipolare i quaternion. Dato il livello, non andiamo troppo a fondo: accontentiamoci che il ped vola nella direzione in cui era già rivolto. (Nei mod reali, il volo è uno degli aspetti più raffinati: *Psychokinetic mod* ad esempio offre 3 modalità di volo con controlli diversi ([Implementare poteri sovraumani GTA V.pdf](#))).

Atterraggio: disattivando il volo a mezz’aria, il player ricade. Eventualmente potremmo applicare una forza verso il basso per farlo scendere più rapidamente, o attivare un’animazione di “atterraggio supereroe” (come la famosa posa inginocchiata). Una chicca: il mod **Iron Man** di JulioNIB, quando Iron Man atterra dopo il volo, genera un impatto che spacca il terreno (ground slam) ([Implementare poteri sovraumani GTA V.pdf](#)), con tanto di effetto particellare e *shockwave*. Potremmo usare un piccolo `World.AddExplosion` al momento dell’atterraggio per simularlo (tipo `ExplosionType.ShockWave`). Ricordate però di farlo solo se l’atterraggio è da una certa altezza o velocità, per non avere esplosioni ad ogni saltello.

Ora che sappiamo volare, continuiamo con i poteri fisici: la **super forza**.

Forza sovrumana (super forza e colpi devastanti)

La super forza si manifesta in vari modi: pugni che mandano i nemici a volare, capacità di sollevare o lanciare oggetti pesanti, atterraggi devastanti, ecc. Non esiste un attributo diretto “forza” per il ped, ma possiamo simulare gli effetti con diverse tecniche:

- **Aumentare i danni del melee:** Possiamo far sì che ogni colpo a mani nude del player sia letale o sbalzi via gli avversari. GTA V non espone direttamente un parametro di danno per i pugni, ma possiamo intervenire in due modi: (a) rilevare quando un ped è colpito e

applicargli danno extra, (b) usare esplosioni invisibili per generare una spinta. Un trucco semplice: quando il player esegue un attacco melee, creiamo un’esplosione di tipo *shove* (spinta) sul punto di impatto, in modo da far volare via il malcapitato. Il tipo di esplosione **ExplosionType.ShockWave** potrebbe essere indicato, con forza bassa ma sufficiente a scaraventare. Ad esempio, nell’evento Tick, potremmo controllare se un ped vicino ha subito di recente danno dal player e, se sì, eseguire `World.AddExplosion(ped.Position, ExplosionType.ShockWave, 1.0f, 0.5f, true, false)`. Questo richiede però di sapere quando un colpo va a segno – non c’è un evento facile, bisognerebbe monitorare la salute dei nemici o usare `HasBeenDamagedBy()` ([Class Ped | Community Script Hook V .NET](#)) ([Class Ped | Community Script Hook V .NET](#)). Un approccio più semplice: creare manualmente l’effetto all’attacco, come segue.

- **Attacco a terra (Ground Slam):** Possiamo implementare un colpo ad area devastante. Ad esempio, premendo un tasto il player esegue un pugno a terra che produce un’onda d’urto tutt’intorno, spazzando via oggetti e ped nelle vicinanze. Questo si fa generando un’esplosione non letale centrata sul player. Esempio:

```
// Ground slam - esplosione intorno al player che butta via oggetti
World.AddExplosion(Game.Player.Character.Position, ExplosionTypeGrenade,
5.0f, 1.0f, true, false);
```

In questo caso usiamo un’esplosione di tipo granata con raggio 5.0 e danno 1.0, udibile (isAudible=true) ma invisibile (isInvisible=false) ([Implementare poteri sovraumani GTA V.pdf](#)). Ciò genererà un’onda d’urto con suono ma poco fuoco, spingendo via entità vicine. Per un’onda d’urto pura senza fuoco, si può usare ExplosionType “**StunGrenade**” e isInvisible true ([Implementare poteri sovraumani GTA V.pdf](#)).

- **Solleva e lancia veicoli:** Un eroe forzuto dovrebbe poter ribaltare o lanciare auto. Questo si può fare applicando forze agli oggetti. Ad esempio, per lanciare un veicolo davanti a noi: puntiamo un veicolo con un raycast o prendiamo il più vicino, poi facciamo `vehicle.ApplyForce(player.ForwardVector * 50f + Vector3.WorldUp * 20f)`. Ciò spingerà il veicolo in avanti e verso l’alto. Per sollevarlo staticamente, potremmo attaccarlo “alla mano” del player usando `vehicle.AttachTo(entityBone)`, ma questo entra nella telecinesi (che affronteremo dopo). Per ora, come demo di forza, potremmo fare che premendo un tasto vicino a un veicolo, questo viene scagliato via.

Codice di esempio: pugno devastante e ground slam:

Supponiamo di attivare la super forza con un tasto (es. **X**). Quando attiva, ogni pressione del tasto attacco (click sinistro) in melee innesca un’esplosione sul target, e premendo un altro tasto (es. **R**) eseguiamo un ground slam AOE.

```
bool superStrength = false;

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.X) {
        superStrength = !superStrength;
        UI.Notify(superStrength ? "Forza sovrumana attivata" : "Forza sovrumana disattivata");
    }
    if(superStrength && e.KeyCode == Keys.R) {
```

```

        // Ground slam: onda d'urto intorno al giocatore
        Vector3 pos = Game.Player.Character.Position;
        World.AddExplosion(pos, ExplosionType.SoundGrenade, 5.0f, 1.0f, true,
true);
    }
}

public void OnTick(object sender, EventArgs e) {
    if(!superStrength) return;
    Ped player = Game.Player.Character;
    // Se il player sta colpendo qualcuno a mani nude
    if(player.IsInCombat && player.Weapons.Current.Hash == WeaponHash.Unarmed) {
        // Trova ped preso di mira (approssimativo)
        Ped target = World.GetClosestPed(player.Position, 5.0f);
        if(target != null && player.IsFacing(target)) {
            // Colpisci il target con un'esplosione invisibile
            World.AddExplosion(target.Position, ExplosionType.ShockWave, 0.5f,
2.0f, false, true);
        }
    }
}

```

Qui abbiamo semplificato: `player.IsInCombat` indica se il player sta menando le mani, e controlliamo `Unarmed` per essere sicuri sia pugno. Prendiamo il ped più vicino entro 5 metri che il player sta guardando (rosso ma sufficiente) e lo “colpiamo” con un’esplosione shockwave invisibile, di raggio piccolo (0.5) e forza 2.0, non letale ma abbastanza forte da scaraventarlo a terra. Per il ground slam con R, usiamo `ExplosionType.SoundGrenade` (che è come una flashbang sonora) con raggio 5: questo farà barcollare tutti intorno e li butta via leggermente ([Implementare poteri sovraumani GTA V.pdf](#)). Potete aumentare il danno se volete che sia letale.

Un supereroe forte potrebbe anche **sollevarsi i nemici**: un modo semplice è afferrare un ped: `ped.AttachTo(player, player.GetBoneIndex(Bone.SK_L_Hand))` per attaccarlo alla mano sinistra del player, poi lanciare con `Detach()` e `ApplyForce`. Ma manipolare attach/detach può causare glitch, quindi a meno di non gestire bene fisica e animazioni risulta complesso. Nel mod **Hulk** di Julio, il personaggio può afferrare un ped e poi lanciarlo; immaginate di implementarlo rilevando un input, congelando il ped (ragdoll) e aggiornandone la posizione rispetto alla posizione della mano del player finché premuto, poi applicando una forza per lanciarlo.

Il nostro scopo è introdurre le idee: la fisica di GTA V ci dà gli strumenti per simulare la super forza con esplosioni e forze applicate. Con questo, passiamo a poteri più soprannaturali e di controllo: **teletrasporto e portali**.

Teletrasporto e portali

Essere in grado di sparire da un luogo e riapparire altrove istantaneamente è un potere affascinante. In GTA V, il teletrasporto è banalmente ottenibile settando le coordinate del giocatore. ScriptHookVDotNet fornisce vari modi per spostare entità:

- `Game.Player.Character.Position = nuovaPosizione;` sposta immediatamente il ped alla posizione specificata.
- Oppure `Game.Player.Teleport(Vector3 destination, bool safeGround = true)` se volessimo un metodo dedicato (non ricordo esista un `Teleport` nel API SHVDN, potremmo dover usare `Position` manualmente).

- Anche i veicoli e altri oggetti possono essere riposizionati settandone Position.

Teletrasporto semplice al marker: Un caso classico è teletrasportare il player al waypoint sulla mappa (il segnalino viola). Molti trainer lo fanno. Possiamo implementarlo con:

```
Vector3 wp = World.GetWaypointPosition();
Game.Player.Character.Position = wp;
```

Eventualmente bisogna trovare un'altezza corretta (Z) sul terreno. Spesso si usa `World.GetGroundHeight(position.X, position.Y)` per ottenere la quota del suolo a quelle coordinate, così da posizionare il ped esattamente a terra e non magari sotto la mappa. Un altro metodo: aggiungere qualche unità a Z e far cadere il ped (il gioco lo adatterà al terreno se non troppo in alto).

Teletrasporto mirato a vista: Un potere tipo “*blink*” a breve distanza o “*teleport where you look*”. Possiamo usare un **raycast** dalla telecamera in avanti per trovare un punto di impatto (su terreno o oggetto) e posizionarci lì. SHVDN offre `World.Raycast(...)` per lanciare un raggio ([GTA 5 The Flash Version 2.0 script mod is head-spinning awesome | PC Gamer](#)). Ad esempio:

```
RaycastResult ray = World.Raycast(GameplayCamera.Position,
GameplayCamera.Direction * 100.0f, IntersectOptions.Everything);
if(ray.DidHitAnything) {
    Game.Player.Character.Position = ray.HitPosition + new Vector3(0,0,1);
}
```

Questo lancia un raggio lungo 100 metri. Se ha colpito qualcosa, prendiamo la posizione di impatto e teletrasportiamo il player lì, aggiungendo 1 metro in altezza per evitare di incastrarci dentro il suolo.

Portali persistenti: Un teletrasporto ancora più interessante è creare due portali stazionari nel mondo, come nel gioco *Portal*. Possiamo implementare un potere che consente al giocatore di piazzare due portali e poi attraversarli. Ad esempio:

- Premere un tasto (es. **T**) per salvare la posizione corrente come *Portale A*.
- Premere di nuovo (o un altro tasto, es. **Y**) per creare il *Portale B*.
- Una volta che entrambi i portali sono posizionati, se il giocatore entra nell'area di uno, viene teletrasportato all'altro.

Dobbiamo quindi:

- Salvare due `Vector3` per gli estremi dei portali.
- Mostrare qualcosa visivamente dove sono i portali (potremmo disegnare un marker cilindrico colorato a terra). Possiamo utilizzare `World.DrawMarker` nel Tick per disegnare, ad esempio, un marker tipo cilindro o anello.
- Controllare la distanza del player da ciascun portale ogni frame. Se è inferiore a una certa soglia (es. 1 metro) e il portale opposto esiste, teleportare il player lì. Bisogna anche evitare teletrasporto immediato di ritorno: introdurre un piccolo ritardo o spostare il player leggermente fuori dal raggio all'uscita.

Per semplicità, assumiamo che il giocatore non porti veicoli attraverso (gestire anche il veicolo è possibile, bisognerebbe teletrasportare l'intero veicolo se il player è in uno).

Codice semplificato per portali:

```

Vector3? portalA = null;
Vector3? portalB = null;
bool justTeleported = false;

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.T) {
        if(portalA == null) {
            portalA = Game.Player.Character.Position;
            UI.Notify("Portale A posizionato");
        } else if(portalB == null) {
            portalB = Game.Player.Character.Position;
            UI.Notify("Portale B posizionato");
        } else {
            // Se entrambi esistono, resettali
            portalA = portalB = null;
            UI.Notify("Portali resettati");
        }
    }
}

public void OnTick(object sender, EventArgs e) {
    // Disegna marker portali se esistono
    if(portalA != null) {
        World.DrawMarker(MarkerType.VerticalCylinder, portalA.Value,
Vector3.Zero, Vector3.Zero,
                           new Vector3(1,1,2), System.Drawing.Color.Blue);
    }
    if(portalB != null) {
        World.DrawMarker(MarkerType.VerticalCylinder, portalB.Value,
Vector3.Zero, Vector3.Zero,
                           new Vector3(1,1,2), System.Drawing.Color.Red);
    }
    if(justTeleported) {
        // piccolo delay per non rientrare subito
        teleportCooldownCounter++;
        if(teleportCooldownCounter > 60) { // ~1 secondo
            justTeleported = false;
            teleportCooldownCounter = 0;
        }
    }
    // Controllo entrata portali
    if(!justTeleported && portalA != null && portalB != null) {
        Ped player = Game.Player.Character;
        if(player.Position.DistanceTo(portalA.Value) < 1.0f) {
            player.Position = portalB.Value + new Vector3(0,0,1);
            justTeleported = true;
        } else if(player.Position.DistanceTo(portalB.Value) < 1.0f) {
            player.Position = portalA.Value + new Vector3(0,0,1);
            justTeleported = true;
        }
    }
}
}

```

In questo esempio, usiamo il tasto **T** per ciclicamente impostare portalA, portalB o resettare entrambi (così con due pressioni di T piazziamo A e B). Usiamo `World.DrawMarker` per disegnare un cilindro verticale (tipo un cerchio a terra) di colore blu per A e rosso per B, così li vediamo ([Portals - GTA5-Mods.com](#)). Nel Tick, se il player entra entro 1 unità di distanza da uno dei due, lo spostiamo all'altro aggiungendo +1 all'altezza (così apparirà leggermente sopra l'uscita per poi cadere a terra). Variabile `justTeleported` serve per evitare che subito dopo il teletrasporto, trovandosi su B, il player venga immediatamente retrasmesso ad A. Impostiamo

quindi `justTeleported = true` e usiamo un piccolo contatore per dare ~1 secondo di immunità al ri-teletrasporto.

([Portals - GTA5-Mods.com](#)) Un semplice marker cilindrico può rappresentare un portale piazzato sul terreno. In questo mod “Portals” i portali sono visibili come cilindri colorati, e teleportano il giocatore da uno all’altro ([Portals - GTA5-Mods.com](#)).

Questa implementazione di base può essere arricchita: ad esempio, attivare i portali solo quando il player preme un tasto specifico entrando (come aprire un varco), oppure permettere di teletrasportare anche i veicoli (come fa la mod *Portals* su GTA5-Mods ([Portals - GTA5-Mods.com](#))). La mod citata consente di creare portali multipli salvati su file e mostra marker permanenti in gioco. Nel nostro contesto, due portali temporanei bastano a illustrare il concetto.

Manipolazione del tempo (bullet-time e freeze temporale)

Il controllo del tempo è un potere meno visibile ma molto potente: possiamo rallentare il mondo, accelerarlo, o addirittura fermarlo. GTA V implementa di default alcuni effetti di slow-motion (nelle missioni o il *special ability* di Michael). Vediamo come replicarli e spingerci oltre.

Slow Motion (Bullet-Time): Come già accennato, la funzione nativa `SET_TIME_SCALE(float scale)` permette di accelerare (>1.0) o rallentare (<1.0) il tempo di gioco ([Implementare poteri sovraumani GTA V.pdf](#)). Usarla è semplice; ad esempio per entrare in bullet-time premendo un tasto (es. **B**):

```
bool timeSlow = false;  
...  
if(e.KeyCode == Keys.B) {  
    timeSlow = !timeSlow;  
    Function.Call(Hash.SET_TIME_SCALE, timeSlow ? 0.2f : 1.0f);  
    UI.Notify(timeSlow ? "Tempo rallentato" : "Tempo normale");  
}
```

Con `scale = 0.2`, tutto il gioco va a 20% della velocità normale: proiettili lenti, auto lente, fisica rallentata, suoni ovattati. Il giocatore incluso. Questo è ottimo per effetti cinematografici. Possiamo combinarlo con super velocità come fatto per Flash, ma consideriamolo come potere a sé stante: ad esempio, un eroe che può **fermare il tempo**.

Freeze totale: Impostare `time scale` a 0.0 teoricamente congelerebbe il mondo – in pratica può mandare in stallo la fisica e l’engine, quindi è sconsigliato. Meglio un valore piccolissimo (0.01) per simulare tempo quasi fermo. Tuttavia, in *time freeze* il giocatore sarebbe anche congelato. Se vogliamo l’effetto che **solo il nostro eroe si muove mentre tutto è fermo** (come Quicksilver in X-Men, o Dio Brando in JoJo “The World” 😊), dobbiamo fare un trucco: rallentare tantissimo il gioco e contemporaneamente potenziare enormemente la velocità del player per compensare. Ad esempio:

```
Function.Call(Hash.SET_TIME_SCALE, 0.05f); // 5% tempo  
Function.Call(Hash.SET_RUN_SPRINT_MULTIPLIER_FOR_PLAYER, Game.Player, 20.0f);
```

Qui riduciamo il tempo a 5% e moltiplichiamo la velocità di movimento del player di 20x. Così il mondo è praticamente fermo, ma il player può muoversi (anche se con animazioni un po’ scattose) quasi a velocità normale relativa. Questo è un hack, con diverse limitazioni, ma è l’unico modo di avere il player non bloccato dal time freeze totale.

Nel caso di *time manipulation* come potere, potremmo presentarlo come “attiva bullet-time” con un tasto (come sopra). Alcune mod implementano addirittura la **retromarcia temporale**, ma questo è fuori dalla portata (richiederebbe salvare lo stato del mondo e riapplicarlo all’indietro).

Rallentare il tempo localmente: Non c’è modo di rallentare solo certe entità (es. solo i nemici ma non il player) senza heavy scripting. In teoria, uno potrebbe mettere in pausa la fisica di ogni ped/veicolo (es. `ped.FreezePosition = true`) e aggiornare manualmente, ma è complesso e oltre gli scopi. Quindi useremo l’approccio global time scale.

Esempio di bullet-time semplice:

Attiviamo con il tasto **B** un rallentatore del tempo. Combiniamolo con un effetto visivo: ad esempio, possiamo applicare un filtro grafico di GTA (ce ne sono, ma SHVDN non li espone facilmente) oppure cambiare leggermente il campo visivo per enfatizzare. Un trucco: far comparire un effetto particolare statico (tipo scintille congelate) – complicato. Manteniamoci sul semplice:

```
bool bulletTime = false;
...
if(e.KeyCode == Keys.B) {
    bulletTime = !bulletTime;
    if(bulletTime) {
        Function.Call(Hash.SET_TIME_SCALE, 0.2f);
        // esempio: attiviamo anche l'abilità speciale di Michael se volessimo effetto colore
        // Function.Call(Hash._START_SCREEN_EFFECT, "FocusIn", 0, false);
    } else {
        Function.Call(Hash.SET_TIME_SCALE, 1.0f);
        // Function.Call(Hash._STOP_SCREEN_EFFECT, "FocusIn");
    }
    UI.Notify(bulletTime ? "Bullet-time ON" : "Bullet-time OFF");
}
```

Questo toglierà il timescale tra normale e 20%. Se volessimo un effetto progressivo (rallentare piano piano e non istantaneo), dovremmo interpolare il valore magari su un paio di secondi – fattibile con una coroutine che incrementa 0.2->1.0 a step.

Curiosità: Nel mod menzionato prima, **Flash mod**, l’auto slow-motion durante la corsa era una opzione configurabile (di default off ([GTA X Scripting - JulioNIB mods: GTA V - New The Flash script mod v2 - NIBStyle](#))). Alcuni mod menu, come il *Simple Trainer*, offrono un “Slow Motion Mode” che fa esattamente questo. Il nostro scopo qui è principalmente mostrare l’uso di `SET_TIME_SCALE`.

Riassumendo: manipolare il tempo in GTA V è semplice dal punto di vista tecnico (una funzione call) ma va usato con attenzione per non rovinare l’esperienza (troppo slowmo prolungato potrebbe buggare AI in script di missioni, ecc.). Raccomandazione: resettare sempre a 1.0 quando il potere finisce, anche in caso di errori, per evitare di lasciare il gioco in slow motion permanente (magari implementare `Script.Aborted` evento per ripristino).

Invisibilità (sparire alla vista)

L’invisibilità del personaggio può essere implementata in modo triviale: la classe `Entity` ha la proprietà `IsVisible`. Basta impostare `Game.Player.Character.IsVisible = false` per far sparire il modello 3D del playe ([Implementare poteri sovraumani GTA V.pdf](#)). Reimpostandola a true, il personaggio ricompare. Questo non influenza la fisica (continua ad

esserci, solo che è invisibile) né i ped NPC (continueranno a inseguirti se eri in allerta). Se l'intento è un potere di **stealth**, bisognerebbe anche agire sulle AI dei nemici per ignorare il giocatore invisibile. Ad esempio, si potrebbe usare `playerPed.IsInvincible = true` (così non possono farti danno) e cambiare il **RelationshipGroup** del player temporaneamente per non essere considerato ostile ([Implementare poteri sovraumani GTA V.pdf](#)). Questo però potrebbe farli smettere di cercarti. Una semplificazione: finché invisibile, setta `playerPed.IsPositionFrozen = true` e teletrasportalo dietro i nemici per fare stealth kill... No, andiamo sul semplice.

Esempio: tasto I per toggle invisibilità.

```
bool invisible = false;
...
if(e.KeyCode == Keys.I) {
    invisible = !invisible;
    Ped player = Game.Player.Character;
    player.Visible = !invisible;
    if(invisible) {
        player.Invincible = true; // magari invincibile mentre invisibile
        UI.Notify("Sei invisibile");
    } else {
        player.Invincible = false;
        UI.Notify("Sei tornato visibile");
    }
}
```

Questo fa scomparire il modello. Notate: le armi che il player impugna potrebbero rimanere visibili in aria! Per evitarlo, potete nascondere anche l'arma equipaggiata o forzare il cambio ad un oggetto invisibile. Oppure, utilizzare la *mimetica totale* che fa anche le armi invisibili (forse `playerPed.Alpha = 0` rende anche l'arma trasparente? `Alpha` controlla la trasparenza dell'entità intera in alcuni casi). Un altro effetto collaterale: se si entra in un veicolo invisibili, il veicolo rimane visibile (quindi sembrerà che il veicolo va da solo).

In mod avanzati, l'invisibilità può essere combinata con effetti sonori (un leggero *whoosh* quando attivi/disattivi) o grafici (sfocatura tipo Predator). C'è un nativo `SET_ENTITY_VISIBLE(entity, bool)` che facciamo indirettamente, e anche la possibilità di rendere invisibili ped e veicoli attorno (ma non serve di solito).

In conclusione, questo è uno dei poteri più facili da ottenere – un singolo flag – ma potenzialmente utile per stealth.

Scudi energetici (difesa e invulnerabilità temporanea)

Un *energy shield* protegge il nostro eroe da proiettili e danni, e magari respinge gli attacchi. Possiamo implementarlo come una combinazione di invulnerabilità e, volendo, un effetto visivo di scudo.

Invulnerabilità temporanea: la proprietà `Ped.IsInvincible` rende il ped immune a danni fisici ([Class Ped | Community Script Hook V .NET](#)). Attivandola, nulla può ridurre la salute del giocatore. Tuttavia, **non** protegge da effetti come ragdoll (cadute) o spostamenti dovuti a urti. Per un vero scudo, potremmo voler anche evitare che il player venga buttato a terra. Potremmo impostare `playerPed.CanRagdoll = false` per evitare il ragdoll. Inoltre, bloccare i proiettili: non c'è un modo immediato di fermare i proiettili in aria senza modelli; ma essendo

invincibile, i proiettili colpiranno ma non faranno danno. Se volessimo, potremmo rimuovere del tutto i proiettili vicini: c'è il metodo `World.GetAllProjectiles()` in SHVDN? Non credo direttamente. In nativo esiste enumerare proiettili, ma è complicato. Per semplicità, ci accontentiamo dell'invincibilità.

Effetto visivo dello scudo: Possiamo creare un oggetto sferico o un'**aura** attorno al player. Ad esempio, GTA V ha un oggetto di collisione chiamato *force field*? Non proprio, ma potremmo spawnerne un'entità invisibile e attivare una collisione particolare. Più facile: usare un **Particle Effect** di tipo scudo (forse esiste qualcosa tipo la bolla del *telefono di Tron*). In mancanza, potremmo generare scintille quando colpiti per far vedere che c'è uno scudo attivo.

Per ora, implementiamo la logica: tasto U per attivare uno scudo per, diciamo, 10 secondi, dopo di che si disattiva e va in cooldown di 5 secondi prima di riutilizzarlo (per dimostrare come gestire durata e cooldown di un potere).

```
bool shieldActive = false;
int shieldTimer = 0;
int shieldCooldown = 0;

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.U && !shieldActive && shieldCooldown == 0) {
        // Attiva scudo per 10 secondi
        shieldActive = true;
        shieldTimer = 600; // 600 tick ~ 10 secondi se 60 fps
        Ped player = Game.Player.Character;
        player.IsInvincible = true;
        player.CanRagdoll = false;
        UI.Notify("Scudo energetico attivato!");
        // Eventuale effetto visivo di attivazione qui
    }
}

public void OnTick(object sender, EventArgs e) {
    if(shieldActive) {
        // Potremmo disegnare un semicerchio attorno al player come indicatore
        // Decrementa timer
        shieldTimer--;
        if(shieldTimer <= 0) {
            // Scudo finito
            shieldActive = false;
            shieldCooldown = 300; // 5 secondi cooldown
            Ped player = Game.Player.Character;
            player.IsInvincible = false;
            player.CanRagdoll = true;
            UI.Notify("Scudo esaurito...");
            // Effetto disattivazione
        } else {
            // Esempio: disegna un piccolo cerchio blu ai piedi del player come
            aura
            World.DrawMarker(MarkerType.DebugSphere, player.Position,
Vector3.Zero, Vector3.Zero,
                               new Vector3(2,2,2), System.Drawing.Color.Cyan);
        }
    }
    if(shieldCooldown > 0) {
        shieldCooldown--;
        if(shieldCooldown == 0) {
            UI.Notify("Scudo ricaricato e pronto.");
        }
    }
}
```

}

In questo codice, lo scudo dura 10 secondi e poi si scarica (potete anche mostrare un timer a schermo, ma qui usiamo Notify per feedback). Durante lo scudo, disegniamo ogni frame una sfera di debug (marker debugSphere) attorno al player con raggio 2 e colore ciano, simulando un campo di forza visibile. In un progetto reale, usereste un effetto migliore (ad esempio c'è un effetto particolare denominato "scr_explosion_shield" o simili utilizzato in alcune missioni, che crea una bolla energetica – bisognerebbe cercarlo nella Native DB).

Lo scudo rende il player invulnerabile e non fa andare in ragdoll, quindi anche se investito da un'auto, dovrebbe restare in piedi. Non blocchiamo fisicamente l'auto però: per far sì che il player sia anche immobile, potremmo aggiungere `player.PhysicsFreezePosition = true`, ma allora non potrebbe muoversi nemmeno lui. Alternativamente, potremmo riflettere i danni: ad esempio, raccogliere se è stato colpito da un proiettile e rimbalzarlo al mittente (avanzato: usare l'evento `Entity.HasBeenDamagedBy(weapon)` e spawnare un proiettile verso l'attaccante). Non entro in questo, però è un'idea divertente per migliorare il potere.

Al termine, impostiamo un cooldown prima di poterlo riattivare, per mostrare come gestire ricarica di poteri – concetto utile se state creando un mod bilanciato (per non spammare scudo continuamente).

Telecinesi (sollevare e lanciare oggetti con la mente)

La **telecinesi** permette al nostro eroe di controllare oggetti e persone a distanza: sollevarli, muoverli, scagliarli. Questo è più complesso perché richiede interagire con altre entità in modi non convenzionali.

Ci sono diversi aspetti:

- **Afferrare un oggetto/ped:** possiamo utilizzare il metodo di attacco `AttachTo`. Ad esempio, per prendere un oggetto, attacchiamo quell'entità al nostro player o a un oggetto invisibile che controlliamo.
- **Muoverlo in aria:** una volta attaccato, l'oggetto seguirà i movimenti del “padre”. Possiamo quindi spostare il padre (es. un marker invisibile) con input per spostare l'oggetto. In alternativa, non attaccare ma continuamente settare la posizione dell'oggetto per farlo inseguire un punto.
- **Rilasciare/lanciare:** staccare l'attach e applicare una forza.

Un'implementazione *basic* telecinesi per un solo oggetto:

- Premendo un tasto (es. E) lanciamo un *raycast* dal player in avanti per trovare l'oggetto più vicino (un veicolo o ped).
- Se trovato, lo “catturiamo”: potremmo congelarlo (`entity.IsPositionFrozen = true` per farlo levitare) e salvarlo come `heldEntity`.
- Finché teniamo premuto E, possiamo magari usare WASD per spostarlo (simile al volo controllato ma applicato all'entità). Oppure la telecamera: farlo orbitare attorno a noi.
- Al rilascio di E, rilasciamo l'entità: togliamo il freeze e se vogliamo lanciarla, applichiamo `ApplyForce` verso la direzione desiderata.

L'**omni telecinesi** (come Magneto che solleva tutto attorno) è ancora più complessa: il mod *Psychokinetic* mostra che farlo su area larga causa drop di fps e andrebbe limitat ([Psychokinetic \[W.I.P\] - GTA5-Mods.com](#)) . Nel nostro caso, facciamo un singolo oggetto per volta.

Codice semplificato: telecinesi su oggetto singolo davanti a noi:

```
Entity heldEntity = null;
float holdDistance = 5.0f;

public void OnKeyDown(object sender, EventArgs e) {
    if(e.KeyCode == Keys.E) {
        if(heldEntity == null) {
            // Tentativo di afferrare oggetto/ped davanti a noi
            RaycastResult ray = World.Raycast(GameplayCamera.Position,
                                                GameplayCamera.Direction * 10f,
                                                IntersectOptions.Everything,
                                                Game.Player.Character);
            if(ray.DidHitEntity) {
                heldEntity = ray.HitEntity;
                heldEntity.IsPositionFrozen = true;
                // Optionally, make ped ragdoll so it doesn't run away
                if(heldEntity is Ped) ((Ped)heldEntity).Ragdoll();
            }
        } else {
            // Se stiamo già tenendo qualcosa e teniamo premuto E, aggiorna
            posizione verso la camera
            Vector3 targetPos = Game.Player.Character.Position +
GameplayCamera.Direction * holdDistance;
            heldEntity.Position = Vector3.Lerp(heldEntity.Position, targetPos,
0.2f);
        }
    }
}

public void OnKeyUp(object sender, EventArgs e) {
    if(e.KeyCode == Keys.E && heldEntity != null) {
        // Rilascia e lancia l'oggetto verso dove guardiamo
        heldEntity.IsPositionFrozen = false;
        Vector3 throwDir = GameplayCamera.Direction;
        heldEntity.ApplyForce(throwDir * 30f);
        heldEntity = null;
    }
}
```

Spiegazione: quando premiamo E iniziamo un raycast di 10 metri in avanti (ignorando il player stesso). Se colpisce un'entità, la blocchiamo (**IsPositionFrozen = true**) e la segniamo come presa. Finché E è tenuto, nel KeyDown continuiamo a posizionare quell'entità davanti a noi a distanza **holdDistance** (che potremmo aumentare/diminuire con rotellina magari). Usiamo una interpolazione lineare (**Lerp**) per muoverla dolcemente verso la posizione desiderata invece di teletrasportarla di colpo, altrimenti potrebbe scattar ([Implementare poteri sovraumani GTA V.pdf](#)) . Quando rilasciamo E, togliamo il freeze e applichiamo una forza in direzione dello sguardo, lanciandola. Questo funziona su oggetti e anche ped (che in GTA sono subclass di Entity). Nel caso di ped, li mettiamo in ragdoll (**Ped.Ragdoll()** li fa afflosciare) così mentre li teniamo non cercano di scappare o combatter ([Implementare poteri sovraumani GTA V.pdf](#)) .

Limiti: se **heldEntity** è un veicolo con un personaggio dentro, meglio prima svuotarlo (il guidatore scenderà comunque se lo muoviamo troppo). Se è troppo pesante (es. un camion), la forza 30f potrebbe sembrare poca, aumentare se serve. Inoltre, a volte **IsPositionFrozen** su ped li fa

comportare stranamente all'atto di rilascio (possono morire sul colpo se rimasti congelati in aria troppo a lungo). Una alternativa è attaccarli a un oggetto invisibile invece di freeze – ma questo è avanzato.

Il mod **Telekinesis 1.0** di *jedijosh920* aveva una meccanica simile, con tasti NumPad per spostare entità su/giù/avanti e lanciar ([Telekinesis - GTA5-Mods.com](#)) ([Telekinesis - GTA5-Mods.com](#)) . La nostra versione è molto basilare, ma dimostra l'uso di raycast e manipolazione diretta di entità.

Controllo mentale (dominare NPC alleati)

Con poteri psichici avanzati, il nostro eroe può non solo spostare oggetti, ma anche controllare la mente dei nemici, facendoli combattere al suo fianco. In GTA V questo si traduce in manipolare l'**AI dei ped**. Possiamo prendere un ped ostile e cambiarlo di fazione.

GTA V ha il concetto di **Relationship Groups**: ped appartenenti a gruppi (gang, polizia, ecc.) che determinano ostilità o alleanza tra loro. Il giocatore di default è nel gruppo "PLAYER". Se vogliamo che un nemico diventi nostro alleato, possiamo assegnarlo al gruppo PLAYER. Inoltre, dobbiamo dargli un compito nuovo: ad esempio attaccare i suoi ex amici.

Approccio: quando il potere di controllo mentale è attivo e il player punta un NPC (magari usando un tasto come indicatore), quell'NPC cambia schieramento e attacca i suoi alleati.

Possiamo usare:

```
enemy.RelationshipGroup = Game.Player.Character.RelationshipGroup;
enemy.Task.ClearAll();
enemy.Task.FightAgainstHatedTargets(50f);
```

Questo codice (dalla nostra guida precedente) assegna il ped al gruppo del player e poi gli dà il compito di combattere i suoi bersagli odiati entro 50 metri ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) . Poiché ora il suo gruppo è PLAYER, i suoi bersagli odiati saranno i nemici del player (cioè quelli che prima erano i suoi alleati, ora diventati ostili a lui). In altre parole, lo abbiamo convertito.

Esempio rapido: tasto **H** per “assoggetta” il ped che hai di fronte.

```
public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.H) {
        Ped target = World.GetClosestPed(Game.Player.Character.Position, 10f,
                                         PedType.Cop, PedType.Civmale,
                                         PedType.Civfemale);
        if(target != null && target.IsAlive && target.IsHuman) {
            target.Task.ClearAll();
            int playerGroup = Game.Player.Character.RelationshipGroup;
            target.RelationshipGroup = playerGroup;
            target.Task.FightAgainstHatedTargets(100.0f);
            UI.Notify($"{target.FullName} ora combatte per te!");
            // Magari aumentiamo anche la sua resistenza
            target.MaxHealth += 100;
            target.Health = target.MaxHealth;
            // E diamogli un'arma
            target.Weapons.Give(WeaponHash.AssaultRifle, 120, true, true);
        }
    }
}
```

Questo cercherà il ped più vicino entro 10 metri (potreste migliorare scegliendo quello mirato con un raycast) e se ne trova uno umano vivo, lo converte: cancella i suoi compiti (smette di sparare a noi), cambia gruppo, e gli assegna il compito di combattere i suoi nemici. Gli diamo anche un bonus alla vita e un fucile per essere utile. Ora sarà nostro alleato (non ci attaccherà e attaccherà chi ci attacca).

Ovviamente, questo ped controllato mentalmente rimarrà così finché vivo o finché la sessione dura. Non tornerà normale perché abbiamo cambiato permanentemente il suo RelationshipGroup. Potremmo salvarlo in una lista se volessimo “liberarlo” in seguito assegnandolo di nuovo al gruppo originale (difficile sapere quale fosse). In un contesto di gioco, potete decidere che il controllo dura X secondi, poi il ped muore o scappa.

Questa tecnica di manipolare AI è molto potente. Pensate che potremmo anche prendere il controllo *diretto* di un ped: esiste `Game.Player.ChangeModel(Model newModel)` che cambia il player in un altro modello. Volendo, avremmo potuto fare **body swap**: trasformare il player nel ped bersaglio, e viceversa, simulando controllo mentale complet ([Implementare poteri sovraumani GTA V.pdf](#)) . Ma questo è estremo: perderemmo il nostro corpo originale a meno di salvarlo e poi ripristinarlo (il che è fattibile, per carità). In generale, meglio evitare di cambiare model al volo se non per trasformazioni volute (vedi sezione prossima).

Ora, proseguiamo con un altro potere d’attacco a distanza: **raggi energetici e laser**.

Raggi energetici e laser dagli occhi

Molti supereroi (Superman, Cyclops degli X-Men, Iron Man) hanno la capacità di emettere raggi o laser. Implementare un raggio visuale continuo in GTA V è una delle cose più complesse perché richiede un effetto grafico (particelle o luce) e danni a distanza. Non possiamo qui ricreare per intero un laser perfetto, ma illustreremo come si potrebbe fare.

Particle Effects: GTA V ha un sistema di effetti particellari (PTFX) con vari effetti disponibili (fuoco, scintille, fumo, ecc.). Ad esempio, c’è un effetto chiamato "scr_rcbarry2" che è un raggio alieno usato in una missione psichedelica. JulioNIB nel mod Superman usa effetti particellari per i laser oculari: uno all’origine (gli occhi) e uno sull’impatto per simulare esplosioni di calore ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) . SHVDN non ha wrapper diretto per creazione di PTFX, quindi bisogna usare nativi: `REQUEST_NAMED_PTFX_ASSET` e `START_PARTICLE_FX_NON_LOOPED_ON_ENTITY` ecc. Data la complessità, in questa sede potremmo semplificare usando un **beam** invisibile: ad esempio, disegnando una semplice linea rossa con la funzione di debug.

ScriptHookVDotNet 3 consente di usare `World.DrawLine(start, end, color)` per disegnare una linea nello spazio 3D (utile per debug). Possiamo usarla per il “raggio” visivo (non è luminoso ma è qualcosa). Per il danno, due opzioni:

- Far esplodere le cose colpite (esplosione incendiaria).
- Sottrarre vita al ped colpito progressivamente.

Possiamo combinare raycast e esplosioni: in ogni Tick in cui il laser è attivo, lanciamo un raggio dritto dalla testa del player in avanti. Se colpisce un ped o veicolo, applichiamo danno. Ad esempio:

```
RaycastResult ray = World.Raycast(playerPed.GetBoneCoord(Bone.SKEL_Head),
```

```

playerPed.ForwardVector * 100f,           playerPed.GetBoneCoord(Bone.SKEL_Head) +
                                         IntersectOptions.Everything, playerPed);
if(ray.DitHitEntity) {
    Entity hitEnt = ray.HitEntity;
    if(hitEnt is Ped) ((Ped)hitEnt).Health -= 20; // toglie 20 hp per tick
    World.AddExplosion(ray.HitPosition, ExplosionType.SmallExplosion, 0.1f,
0.0f, false, true);
}

```

Questo lancia un raggio dagli occhi del player (bone head) in avanti 100m. Se colpisce qualcosa, se è ped togliamo salute, e creiamo una piccola esplosione *visiva* (tipo fuoco piccolo) per mostrare l'impatto ([Implementare poteri sovraumani GTA V.pdf](#)). L'esplosione di tipo SmallExplosion con danno 0.1 e raggio 0 causa un po' di fuoco e fumo localizzato senza sbalzare troppo l'entità (usando isAudible false e isVisible true avremmo un impatto silenzioso visibile solo con fuoco). Per migliorare, potremmo anche incendiare l'entità colpita: `hitEnt.IsOnFire = true`.

Attivazione del laser: per esempio, potremmo richiedere di tenere premuto tasto destro del mouse (mira) e click sinistro per sparare il laser dagli occhi. Oppure un tasto toggle continua. Dato che questo è uno dei poteri più “avanzati”, lo collochiamo come se l'eroe lo attivasse con **L** per un raggio continuo finché tiene premuto.

Codice esempio ridotto:

```

bool laserOn = false;

public void OnKeyDown(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.L) {
        laserOn = true;
    }
}

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.L) {
        laserOn = false;
    }
}

public void OnTick(object sender, EventArgs e) {
    if(laserOn) {
        Ped player = Game.Player.Character;
        Vector3 origin = player.GetBoneCoord(Bone.SKEL_Head); // posizione testa
        Vector3 target = origin + player.ForwardVector * 100f;
        // Disegna raggio rosso
        World.DrawLine(origin, target, System.Drawing.Color.Red);
        // Raycast per colpire
        RaycastResult ray = World.Raycast(origin, target,
IntersectOptions.Everything, player);
        if(ray.DitHitEntity) {
            Entity ent = ray.HitEntity;
            World.AddExplosion(ray.HitPosition, ExplosionType.SmallExplosion,
0.0f, 0.1f, false, false);
            if(ent is Ped) {
                Ped p = (Ped)ent;
                p.Health -= 10;
                p.IsOnFire = true;
            } else if(ent is Vehicle) {
                Vehicle v = (Vehicle)ent;
                v.Health -= 50;
                v.EngineHealth -= 50;
            }
        }
    }
}

```

```
        }
    }
}
```

Qui ogni frame col laser attivo disegniamo la linea e infliggiamo danno. Notare: stiamo togliendo vita direttamente, il che potrebbe non uccidere ped protetti da armatura (bisognerebbe fare p.Armor se hanno). Mettiamo anche a fuoco i ped, e danneggiamo i veicoli (EngineHealth per farli esplodere dopo un po').

Questo laser è semplificato e non ha un vero effetto grafico scintillante, ma il concetto c'è. In un contesto reale, si vorrebbe usare i particellari: **request** dell'effetto ("core" rosso tra gli occhi), **start** effetto loopato verso il target, etc. Non entriamo nel dettaglio codice PTFX, ma ecco un'idea: c'è un nativo **START_NETWORKED_PARTICLE_FX_NON_LOOPED_ON_ENTITY_BONE** dove puoi specificare un bone (ad esempio, ossa degli occhi). Julio nei suoi mod ha spesso file .ini o codice con i nomi di particelle, per esempio dice: *"laser occhi che bruciano i bersagli e fanno esplodere le gomme"* nel suo mod Superma ([Implementare poteri sovraumani GTA V.pdf](#)) . Il nostro laser già incendia i bersagli; far esplodere le gomme delle auto colpite potremmo aggiungerlo facilmente (iterare su tires e burst).

Con questo chiudiamo i poteri di attacco. Ora affrontiamo l'ultimo grande potere: **trasformazione** del personaggio.

Trasformazioni multiple (cambiare forma o costume)

Alcuni supereroi possono trasformarsi: Bruce Banner diventa Hulk, Tony Stark indossa l'armatura Iron Man, ecc. Nelle mod, questo si ottiene **cambiando il modello 3D** del player o applicando vestiti diversi.

ScriptHookVDotNet fornisce un metodo comodissimo: `Game.Player.ChangeModel(Model newModel) ([Implementare poteri sovraumani GTA V.pdf](#)) . Possiamo caricare un modello (ped) e sostituire il nostro attuale. Ad esempio, se abbiamo installato un ped model aggiuntivo chiamato "HULK", possiamo fare:

```
Model hulk = new Model("HULK");
hulk.Request(500);
while(!hulk.IsLoaded) Script.Wait(100);
Game.Player.ChangeModel(hulk);
Game.Player.Character.Style.SetDefaultClothes(); // vestiti di default per il
ped
hulk.MarkAsNoLongerNeeded();
```

Questo snippet, presente anche nella nostra guida precedente, carica il modello e lo applic ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) . **SetDefaultClothes** è utile per far sì che se il ped ha componenti casuali, gliene applichi di default (evitando che Hulk spunti magari con vestiti strani se il model lo supporta).

Possiamo dunque costruire un potere di trasformazione: ad esempio premendo un tasto cambiamo modello. Potremmo permettere **più trasformazioni** ciclabili: per esempio, premendo un tasto si passa dalla forma umana a Hulk, ripremendo si passa a Iron Man, ripremendo torna umano. Ci basta tenere traccia dello stato corrente.

Attenzione: quando cambiamo modello, perdiamo riferimenti al ped precedente (che viene sostituito). Se avevamo armi, possono essere perse; la posizione resta la stessa. Inoltre, alcuni poteri

attivi andranno disattivati perché legati al ped vecchio (ma se è lo stesso Player non dovrebbe cambiare molto, a parte rifare IsInvincible ecc. sul nuovo ped se serve).

Esempio di trasformazione ciclica: consideriamo 3 forme: **Normale**, **Hulk**, **IronMan**. Dobbiamo avere i model hash o nomi: Hulk supponiamo di averlo come addon ped chiamato "HULK", IronMan magari come "IRONMAN" (bisogna avere questi addon installati, qui è ipotetico).

```
enum Form { Normal, Hulk, IronMan }
Form currentForm = Form.Normal;
Model normalModel = new Model(PedHash.Franklin); // supponiamo che l'eroe
iniziale fosse Franklin

public void OnKeyUp(object sender, KeyEventArgs e) {
    if(e.KeyCode == Keys.T multiply) { // supponiamo tasto * sul numpad per
trasformazione
        if(currentForm == Form.Normal) {
            TransformInto("HULK");
            currentForm = Form.Hulk;
            UI.Notify("Trasformazione in Hulk!");
        } else if(currentForm == Form.Hulk) {
            TransformInto("IRONMAN");
            currentForm = Form.IronMan;
            UI.Notify("Trasformazione in Iron Man!");
            // magari attivare l'armatura (dare armi, abilità volo? Potremmo
farlo qui)
        } else if(currentForm == Form.IronMan) {
            // torna normale
            TransformInto(normalModel);
            currentForm = Form.Normal;
            UI.Notify("Tornato forma normale");
        }
    }
}

private void TransformInto(Model model) {
    Ped player = Game.Player.Character;
    model.Request(500);
    while(!model.IsLoaded) Script.Wait(50);
    Game.Player.ChangeModel(model);
    Game.Player.Character.Style.SetDefaultClothes();
    model.MarkAsNoLongerNeeded();
    // Mantenere alcuni attributi
    Game.Player.Character.Health = player.Health;
    Game.Player.Character.Armor = player.Armor;
}
```

Abbiamo definito una funzione **TransformInto** che può prendere sia un **Model** sia un nome/Hash e fa la trafia di caricamento e cambio. Notiamo che proviamo a mantenere la salute/armatura (prendendo dal ped prima di cambiare). Potremmo anche mantenere posizione (anche se dovrebbe rimanere uguale), armi (dovremmo ri-dare le stesse armi al nuovo ped, perché spesso si resetta l'arsenale). Si potrebbe salvare la lista **player.Weapons** prima del cambio e riapplicarla.

In questo esempio, stiamo usando Franklin come forma “normale” di base (se il giocatore all’inizio era Franklin). In un mod generico, potremmo considerare **normalModel = Game.Player.Character.Model** all’inizio, così funziona con qualsiasi ped di partenza (Michael, Trevor, MP character, etc.).

La trasformazione non è solo estetica: di solito comporta anche diversi poteri correlati. Per esempio, **Hulk** potrebbe automaticamente attivare super forza e super salto, ma disabilitare l'uso di armi da fuoco; **Iron Man** potrebbe attivare volo e raggi (repulsori) ma magari essere più fragile a mele. Sta al design del mod. Il nostro focus è mostrare come cambiare modelli.

Nota su modelli custom: Per usare nomi come "HULK" e "IRONMAN", bisogna averli installati tramite mod (addonpeds). In caso contrario, `new Model("HULK")` fallirà (`IsLoaded` rimarrà false). In un contesto reale, bisogna controllare se il model esiste, e segnalare all'utente di installarlo se no. In questa guida assumiamo di avere i modelli.

Ulteriori trasformazioni: Questo schema può essere esteso a quante forme volete: basta aggiungere enumerazioni e ciclarle. Ad esempio, un mod *Shapeshift* potrebbe passare attraverso animali (coyote, uccello, squalo) caricando quei model (PedHash coyote esiste? Sì, come `PedHash.Coyote`). Bisogna però stare attenti perché cambiare in animale spesso limita i controlli (il gioco ha controlli diversi per animali, es. niente armi, e a volte glitch se poi torni umano). Quindi testare sempre.

Ritorno al modello originale: nel codice sopra, teniamo `normalModel` e lo riutilizziamo. Questo funziona se all'inizio dell'esecuzione lo salviamo. Nel caso di Franklin come `PedHash.Franklin` è facile, ma se l'utente inizia con un ped custom? Meglio fare `normalModel = Game.Player.Character.Model`; nel script Start.

Con questo, abbiamo implementato varie trasformazioni. In un mod completo, si potrebbe combinare questo con un menu per selezionare la forma (es. con NativeUI, un menu “Seleziona personaggio” con opzioni). Ad esempio il mod *JulioNIB's Flash* permette di cambiare tra vari velocisti (Flash, Zoom, ecc.) cambiando semplicemente modello e texture.

Ora che abbiamo dotato il nostro eroe di una vasta gamma di poteri (dalla velocità al volo, dai portali al controllo mentale), è tempo di discutere come mantenere organizzato il codice man mano che si aggiungono abilità, e vedere alcune best practice generali per mod stabili e compatibili.

Best practice di sviluppo mod: manutenzione, HUD e compatibilità

Implementare singoli poteri è stimolante, ma se vogliamo mettere tutto insieme in un'unica “mod supereroe” ci sono delle sfide: evitare codice disordinato, gestire conflitti tra poteri, presentare un’interfaccia utente chiara e assicurarsi che la mod conviva bene con altre mod. In questa sezione riepiloghiamo alcuni consigli e tecniche per portare la vostra mod al livello professionale.

Organizzazione modulare del codice

Man mano che aggiungiamo funzioni, il codice può diventare difficile da gestire. È utile strutturarlo in modo **modulare** e orientato agli oggetti. Ad esempio, possiamo creare una classe base `Power` con metodi virtuali `Activate()`, `Deactivate()`, `OnTick()` e proprietà come ``IsActive`` ([Implementare poteri sovraumani GTA V.pdf](#)) . Per ogni potere (volo, forza, velocità, ecc.) creiamo una classe derivata che implementa questi metodi. Il nostro script principale tiene una lista di tutti i poteri disponibili e li aggiorna ogni fram ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) .

Pseudo-codice di struttura:

```
abstract class PowerBase {
    public bool IsActive;
    public virtual void Activate() { IsActive = true; }
    public virtual void Deactivate() { IsActive = false; }
    public virtual void OnTick() {}
}

class FlightPower : PowerBase {
    public override void Activate() { base.Activate();
Game.Player.Character.HasGravity = false; ... }
    public override void Deactivate() { base.Deactivate();
Game.Player.Character.HasGravity = true; ... }
    public override void OnTick() { if(IsActive) { /* gestisci input volo */ } }
}
// ... analoghe classi per SpeedPower, StrengthPower, ecc.

List<PowerBase> powers = new List<PowerBase>() { flight, speed, strength, ... };

public void OnTick(...) {
    foreach(var p in powers) {
        p.OnTick();
    }
}
```

Così ogni potere incapsula la propria logica e stato. Il main script solo delega e coordina. Ad esempio, se premiamo il tasto di volo, chiamiamo `flight.Activate()`. Possiamo anche fare in modo che quando un potere si attiva, il main disattivi eventuali incompatibili: es. se attivo volo, disattiva super velocità a terra, per non avere conflitti ([Implementare poteri sovraumani GTA V.pdf](#)). Possiamo definire priorità o esclusioni: magari aggiungendo proprietà `MutuallyExclusiveWith` in PowerBase, o gestendolo manualmente in `OnKeyUp` handler (if volo attivato -> if speed.IsActive then speed.Deactivate()) ([Implementare poteri sovraumani GTA V.pdf](#)).

Questo approccio ad oggetti migliora la **manutenibilità ed estendibilità**: aggiungere un nuovo potere richiede solo di creare una nuova classe derivata e integrarla, senza toccare ogni if/else sparso nel codice ([Implementare poteri sovraumani GTA V.pdf](#)).

Gestione dell'interfaccia utente (HUD e menu)

Con tanti poteri, serve un modo per informare il giocatore quali sono attivi, magari permettere di selezionarli o configurarli. Due strade principali: usare librerie di menu come **NativeUI** oppure disegnare elementi custom sullo schermo.

NativeUI è una libreria molto diffusa (originariamente creata da Guadmaz) per creare menu interattivi nello stile di GTA (liste, submenu, ecc.) con poche righe di codice. Molte mod includono il file NativeUI.dll e lo usano per i propri menu di configurazione. Ad esempio, potremmo fare un menu “Poteri” dove attivare/disattivare ciascun potere, oppure selezionare la trasformazione. In questa guida non scriveremo il codice completo di menu, ma è utile sapere che esiste ed è consigliato ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)). Se decidete di usarlo, includete la dll e documentate l’uso (molti utenti di mod sono abituati a premere un tasto come Numpad0 o F5 per aprire menu delle mod, ad esempio).

Un altro esempio con NativeUI: la mod *Starman* (aperta su GitHub) utilizza una **Barra** di durata sullo schermo per un potere di invincibilità, creata con la classe **BarTimerBar** di NativeUI ([Implementare poteri sovraumani GTA V.pdf](#)) . NativeUI infatti fornisce anche elementi HUD come barre e testi facili da posizionare.

Se non volete dipendere da librerie esterne, ScriptHookVDotNet fornisce il namespace **GTA.UI** e metodi come **UI.ShowSubtitle(...)** per testi temporanei, oppure potete disegnare figure con **UI.DrawRect** e `UI.DrawText ([Implementare poteri sovraumani GTA V.pdf](#)) . Con un po' di matematica, potete disegnare sullo schermo icone o scritte per indicare poteri attivi (magari usando il texture dictionary di GTA se volete icone). Questo è più laborioso che usare NativeUI, ma vi dà libertà totale stilistica.

Un'interfaccia popolare nelle mod supereroi è la **Power Wheel**: una ruota selezionabile tenendo premuto un tasto (simile alla ruota delle armi, ma con icone dei poteri). Implementarla significa rilevare il input e disegnare un cerchio con spicchi selezionabili – non banale, ma come concetto potete usare UI.DrawRect/Texture per i segmenti e highlight in base alla posizione del mouse. In Flash mod, JulioNIB implementò una power wheel per selezionare i vari attacchi (tornado, lightning, phase, ecc. ([Implementare poteri sovraumani GTA V.pdf](#)) . Probabilmente internamente ogni potere corrispondeva a una funzione modulare attivata dal men ([Implementare poteri sovraumani GTA V.pdf](#)) .

Per iniziare, magari accontentatevi di hotkey e notifiche su schermo. Poi, se la mod cresce, potrete integrare NativeUI per aggiungere un menu di configurazione (es. tasti, durata poteri, ecc.) e un HUD più carino (icone, barre di energia per cose come lo scudo che si ricarica).

Compatibilità con altre mod e stabilità

Quando distribuite una mod, tenete a mente che l'utente potrebbe averne decine attive insieme. Ecco alcuni punti per minimizzare conflitti:

- **Tasti configurabili:** come già detto, date modo di cambiare gli hotkey. Così se un'altra mod usa lo stesso tasto, l'utente può risolvere. Anche evitare tasti troppo comuni (es. E di default è usato per azioni in GTA, forse meglio usare combinazioni tipo Ctrl+un tasto).
- **Evitare global state persistente:** Non usate ad esempio **World.GravityLevel = X** (che cambia la gravità globale) senza ripristinarla, perché influenzerebbe tutto e magari un'altra mod non lo aspetta. Lo stesso per **TimeScale** – sempre riportarlo a 1.
- **Cleanup di entità:** Se la vostra mod spawna oggetti/ped/veicoli, considerate di eliminarli quando non servono più, specialmente se la mod viene disattivata. Potete implementare **Script.Aborted += ...** evento, o override **Dispose**, per rimuovere eventuali entità ancora presenti quando lo script termina. Ad esempio, se avete creato portali o ped controllati, rimuoveteli o ripristinateli (altrimenti potrebbero rimanere nel mondo permanentemente finché il gioco è aperto, occupando risorse).
- **Threading e Wait:** SHVDN consente di usare **Script.Wait** per far pause. Evitate di fare wait lunghi nel main Tick che possano congelare lo script. Se vi serve un ritardo lungo, meglio fare un piccolo coroutine (es. una funzione async simulation: usare un Timer o incrementare un contatore in tick come abbiamo fatto per il cooldown scudo) piuttosto che **Wait(5000)** dentro Tick che bloccherebbe l'aggiornamento di altre logiche.

- **Versioni di ScriptHookVDotNet:** Mantenetevi aggiornati con l'ultima versione. A volte dopo un update di GTA, le mod vanno ricompilate con le nuove versioni. Seguite i thread su community (ad es. r/GTAV_Mods su Reddit) per scoprire incompatibilità. Se una funzione non va perché deprecata, cercate alternative sul changelog di SHVDN.
- **Performance generale:** Testate la mod in situazioni intense (città trafficata, molte entità). Se notate cali, ottimizzate come discusso. Ad esempio, la telecinesi di massa può calare FPS: potete limitare numero di oggetti influenzati o abbassare frequenza controlli in quei frangenti ([Psychokinetic \[W.I.P\] - GTA5-Mods.com](#))】. Dare la possibilità all'utente di regolare nel config alcune intensità (raggio di azione, numero massimo di ped controllati, ecc.) può aiutare a farla girare su più PC.

Debugging efficace durante lo sviluppo

Durante lo sviluppo della mod, incontrerete bug o comportamenti inaspettati. Ecco alcuni suggerimenti per individuarli e risolverli:

- **Log degli errori:** SHVDN produce un file `ScriptHookVDotNet.log` nella directory di gioco, con traccia degli errori script. Se il vostro codice lancia un'eccezione non gestita, finirà lì. Controllate questo log se “qualcosa non funziona”: spesso troverete uno *stack trace* util ([Implementare poteri sovraumani GTA V.pdf](#))】. Ad esempio, NullReferenceException alla linea X del vostro script vi dice dove intervenire.
- **Try-Catch e notifiche:** Per parti critiche, potete racchiudere in blocchi try-catch e, invece di far crashare lo script, catturare l'errore e magari mostrare `UI.Notify("Errore potere volo: " + ex.Message) ([Implementare poteri sovraumani GTA V.pdf](#))】. Così il gioco continua e voi avete un indizio dell'errore.
- **Stampa di debug:** Utilizzate `UI.Notify` o `Console.WriteLine` (la console di SHVDN) per stampare valori e stati interni mentre testate. Ad esempio, potete notificare “volo attivato, velocità = X” per capire se la logica entra in funzion ([Implementare poteri sovraumani GTA V.pdf](#))】. Ricordatevi di rimuovere o disabilitare questi log nella versione finale, per non appesantire o infastidire l'utente.
- **Console interattiva:** ScriptHookVDotNet ha una console che si apre premendo il tasto **tilde** (se abilitata nelle impostazioni). Da lì potete eseguire comandi e richiamare funzioni dei vostri script in tempo reale ([Implementare poteri sovraumani GTA V.pdf](#))】. Ad esempio, potreste esporre un comando tipo SUPERJUMP che attiva il super salto, e provarlo dalla console. È avanzato, ma utile.
- **Debugger di Visual Studio:** In teoria è possibile attaccare il debugger di VS al processo GTA5.exe e mettere breakpoint nel vostro script, poiché è .NET. Tuttavia, vi serve compilare la DLL col debug symbols e attaccare in tempo. Non sempre funziona fluidamente, ma potete tentare. Più spesso, il metodo di debug “print su schermo” è sufficiente.
- **Test incrementale:** Aggiungete e testate una funzionalità alla volta ([Implementare poteri sovraumani GTA V.pdf](#))】. Se implementate 5 poteri insieme e qualcosa va storto, sarà più arduo capire quale. Ad esempio, prima implementate il volo e testatelo a fondo (in varie situazioni: e se attivo volo in macchina? e se durante il volo attivo super velocità accidentalmente? etc.). Poi passate al potere successivo. Costruire gradualmente aiuta a localizzare bug quando compaiono.

Risorse e supporto della community

Non siete soli: la community di modding ha tante risorse utili:

- **Forum GTA5-Mods, sezione Tutorials/Scripts:** Ci sono guide e discussioni. Ad esempio, esistono thread su come aumentare la velocità di corsa, come usare NativeUI, ecc. Un utente ha chiesto come modificare run speed e qualcuno ha postato un snippet con SET_RUN_SPRINT_MULTIPLIE ([Implementare poteri sovraumani GTA V.pdf](#)) . Cercando, troverete spesso che qualcuno ha già provato ciò che tentate.
- **Reddit r/GTAV_mods:** utile per soluzioni a problemi post-patch e consigli generali ([Implementare poteri sovraumani GTA V.pdf](#)) . Meno tecnico del forum, ma buono per stare aggiornati su ScriptHookV, ScripthookVDotNet e relative *workaround* quando il gioco si aggiornerà ([Implementare poteri sovraumani GTA V.pdf](#)) .
- **Codice sorgente di ScriptHookVDotNet:** Il progetto SHVDN è open source su GitHub. Guardare il codice interno può chiarire comportamenti. Ad esempio, se volete capire esattamente come funziona Ped.ApplyForce, potete vedere che internamente chiama il nativo APPLY_FORCE_TO_ENTITY con certi parametri ([Implementare poteri sovraumani GTA V.pdf](#)) . Capire i dettagli vi aiuta ad usarlo meglio.
- **Mod open-source di esempio:** Alcune mod script hanno il codice disponibile. Abbiamo menzionato *Starman* (fornisce spunti su HUD e invincibilità temporanea ([Implementare poteri sovraumani GTA V.pdf](#)) , c'è anche *Superman Ultimate* su GTA5-Mods con codice su GitHub, e varie su FiveM forum. Studiare codici esistenti è uno dei modi migliori per imparare trucchi e best practice.

Esempi di mod famose e tecniche utilizzate

Per ispirazione, passiamo in rassegna alcune mod di superpoteri già create da modder esperti, evidenziando come implementano certe abilità (anche se il codice non è pubblico, le funzionalità sono documentate):

- **GTA V Iron Man mod** (JulioNIB): Introduce volo propulso, repulsori, missili, e un HUD da armatura. La versione 2.0 migliora animazioni e aggiunge il *ground slam* – l'atterraggio pesante che abbiamo discusso, realizzato probabilmente con un'esplosione all'impatt ([Implementare poteri sovraumani GTA V.pdf](#)) . Include anche una nuova UI stile casco di Iron Man ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) , realizzata disegnando elementi sullo schermo (JulioNIB spesso integra le sue proprie UI). Questa mod evidenzia come combinare più poteri: volo, super forza (solleva auto), armi avanzate, tutto in uno. Richiede anche modelli 3D personalizzati delle armature (forniti a parte).
- **GTA V Thanos mod** (JulioNIB): Ispirata ad Avengers Endgame, implementa i poteri delle Gemme dell'Infinito. Interessanti sono il **teletrasporto tramite portali** e la famigerata “snap” (schiocco di dita) che dissolve metà dei ped. I portali presumibilmente appaiono come cerchi scintillanti (probabilmente particle FX) e funzionano come i nostri ma su scala cinematografica. Lo snap elimina ped creando un effetto dissolvenza in polvere: questo si può ottenere togliendo invincibilità a un ped e riducendone lentamente la vita mentre si emette un particle effect di polveri ([Implementare poteri sovraumani GTA V.pdf](#))

([Implementare poteri sovraumani GTA V.pdf](#)) . Queste idee mostrano uso creativo di meccaniche base (ridurre vita gradualmente, spawnare particelle) per effetti scenografici.

- **Psychokinetic mod** (NModds, WIP): Una delle mod più ambiziose, include telecinesi generica e vari poteri elementali (ghiaccio, fuoco, elettricità ([Implementare poteri sovraumani GTA V.pdf](#)) . Dal suo readme si apprendono combinazioni di tasti: ad esempio *F4* per prendere un ped (telecinesi), *Shift+X / Ctrl+X* per spingerlo o tirarlo, *Shift+E* per lock targe ([Implementare poteri sovraumani GTA V.pdf](#)) . Questo implica un sistema di controlli complesso dove diversi tasti modificatori danno poteri differenti – un ottimo esempio di gestione input avanzata. Implementa anche volo con 3 modalità cambiabili con CapsLoc ([Implementare poteri sovraumani GTA V.pdf](#)) e trasformazione (Shift+T) – probabilmente cambia il modello del playe ([Implementare poteri sovraumani GTA V.pdf](#)) . Nonostante il codice non sia open, la descrizione dettagliata funge da design document pieno di ide ([Implementare poteri sovraumani GTA V.pdf](#)) .
- **Superman mod** (JulioNIB & altri): Ne esistono diverse versioni. Quella di Julio include volo libero, super forza, invulnerabilità, **sparo laser dagli occhi** e **soffio gelido** (che congela i nemici). Il laser è implementato come detto con particelle e danni da fuoc ([Implementare poteri sovraumani GTA V.pdf](#)) ([Implementare poteri sovraumani GTA V.pdf](#)) . Il soffio gelido probabilmente usa un cono di particelle e applica un effetto freeze (forse impostando `ped.Task.Freeze(duration)` o semplicemente trasformando i ped in oggetti ghiacciati – c’è un nativo per “freeze ped prop”? Non certo). Comunque, il mod mostra come aggiungere anche effetti di stato sui nemici (bruciato, congelato).
- **The Flash mod** (JulioNIB): Già trattato, i punti chiave erano velocità aumentata, **wall run** (corsa sui muri, probabilmente realizzata rilevando collisione con parete e applicando una forza opposta alla gravità per rimanere aderenti), **fase attraverso gli oggetti** (set `NoCollision` per player temporaneamente per attraversare ostacoli), e gli attacchi speciali (tornado, punch machine, ecc.). La *tornado hands* ad esempio crea un vortice: ottenibile ruotando velocemente il ped su se stesso e generando venti (esplosioni d’aria) attorno, oppure con particelle. Questi dettagli fanno capire quanta sperimentazione c’è dietro: migliorare animazioni di salto, aggiungere scie, camera shake ad alta velocità, etc.
- **Mods Marvel/DC vari:** Ci sono mod per Hulk (super forza estrema, salti enormi, atterra causando crateri), per Spider-Man (lanciare ragnatele e oscillare tra i palazzi: realizzato probabilmente con attacchi Raycast e Attach rope tra edifici – esiste un nativo per creare cavi, e usare la fisica dei cavi per simulare la ragnatela), per Doctor Strange (crea portali, clonazione astrale, scudi magici – molti FX), per Green Lantern (costruisce oggetti verdi di energia – spawn di oggetti modellati a forma di puño gigante o altro). Ognuna di queste richiede trucchi particolari, ma alla base usano quel che abbiamo imparato: spawn/attach entità, forze fisiche, manipolazione ped AI, e heavy use di particelle e animazioni personalizzate.

Come si può intuire, la differenza tra la nostra mod e quelle di modder veterani sta molto nell’aggiungere **rifiniture**: animazioni ad hoc, modelli speciali, effetti audiovisivi curati. La logica fondamentale dei poteri resta simile a quanto abbiamo implementato.

Conclusione

Seguendo questo tutorial, abbiamo costruito passo dopo passo una serie di poteri sovrumani in GTA V: dal volo alla super velocità, dalla telecinesi al teletrasporto, passando per scudi, invisibilità e trasformazioni. Abbiamo visto come sfruttare le API di ScriptHookVDotNet3 e i nativi di GTA V per piegare le regole del gioco e creare effetti spettacolari.

Il cammino da qui alla realizzazione di una mod completa (*il tuo MarvelScript*) richiede tempo e sperimentazione, ma ora hai le basi e molti esempi pratici su cui modellare il tuo codice. Ricorda di procedere in modo modulare, testare ogni aggiunta, e fare tesoro delle best practice per evitare bug e garantire compatibilità ([Implementare poteri sovraumani GTA V.pdf](#)). Usa la creatività: combina questi poteri, modifica i parametri, aggiungi i tuoi – GTA V è il tuo sandbox di supereroi.

Infine, tieni d'occhio la community per aggiornamenti e nuove idee, e non esitare a studiare e imparare dai migliori. Con dedizione, potrai portare nel gioco i tuoi supereroi (o supercattivi) preferiti, ricreando le loro abilità in modo convincente e divertente. **Buon coding e buon divertimento!**