

## (Πρόχειρο) Writeup για το topDown

Authors: Sizon95, mikev

### Περιγραφή:






Yesterday, I came back home to find a simple note from my brother: “Hey there sis, you think you are a better gamer than me right? Heh, no matter what you do you ‘ll never be able to beat me at this game”.

Please help me prove him wrong!!

**Attachment:** chall\_topDown.zip

### Λύση

Αφού κατεβάσουμε το zip του παιχνιδιού και το κάνουμε extract, βλέπουμε την παρακάτω δομή:

 MonoBleedingEdge	12/2/2024 3:04 μμ	Φάκελος αρχείων	
 topDown_Data	12/2/2024 3:04 μμ	Φάκελος αρχείων	
 topDown.exe	12/2/2024 2:30 μμ	Εφαρμογή	651 KB
 UnityCrashHandler64.exe	12/2/2024 2:30 μμ	Εφαρμογή	1.089 KB
 UnityPlayer.dll	12/2/2024 2:30 μμ	Επέκταση εφαρμο...	30.199 KB

Πολύ γρήγορα καταλαβαίνουμε ότι πρόκειται για Unity Game.

Αρχικά, δοκιμάζουμε να παίξουμε κανονικά το παιχνίδι, το οποίο πρόκειται για ένα generic racing game που πρέπει να μαζεύεις νομίσματα και να αποφεύγεις τα άλλα οχήματα.

Σύντομα όμως, καταλαβαίνουμε ότι το παιχνίδι είναι unbeatable, καθώς ύστερα από κάποιο διάστημα μια μεγάλη συστοιχία αμαξιών εμφανίζονται στην οθόνη, που είναι αδύνατον να προσπεράσουμε:

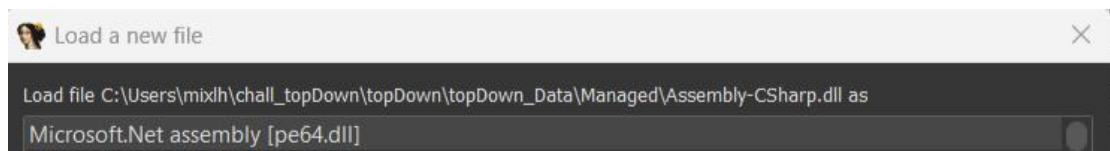


Άρα πρέπει να βρούμε έναν τρόπο να χακάρουμε το παιχνίδι.

Όταν αντιμετωπίζουμε ένα Unity Game, πρέπει να γνωρίζουμε ότι το αρχείο που περιλαμβάνει τον κώδικα της εφαρμογής (σε εκτελέσιμη μορφή) ονομάζεται συνήθως Assembly-CSharp.dll (ή GameAssembly.dll)

Υπάρχει πάντα το ενδεχόμενο ο προγραμματιστής να θέλει να προστατεύσει τον κώδικα του προγράμματός τους, δυσκολεύοντας το reverse engineering, οπότε σε αυτήν την περίπτωση χρησιμοποιεί το εργαλείο il2cpp, το οποίο μετατρέπει τον .NET κώδικα (IL) σε κώδικα c++, ώστε να δημιουργήσει ένα native binary, το οποίο γίνεται reverse πολύ πιο δύσκολα (χρησιμοποιεί την κλασική x86 αρχιτεκτονική) και απαιτεί εργαλεία όπως (IDA, Ghidra, il2cppdumper κτλ).

Ευτυχώς στην περίπτωση μας το Assembly-CSharp.dll (Path: topDown\topDown\_Data\Managed ) και αναγνωρίζεται ως .NET Assembly (π.χ. μέσω της IDA)



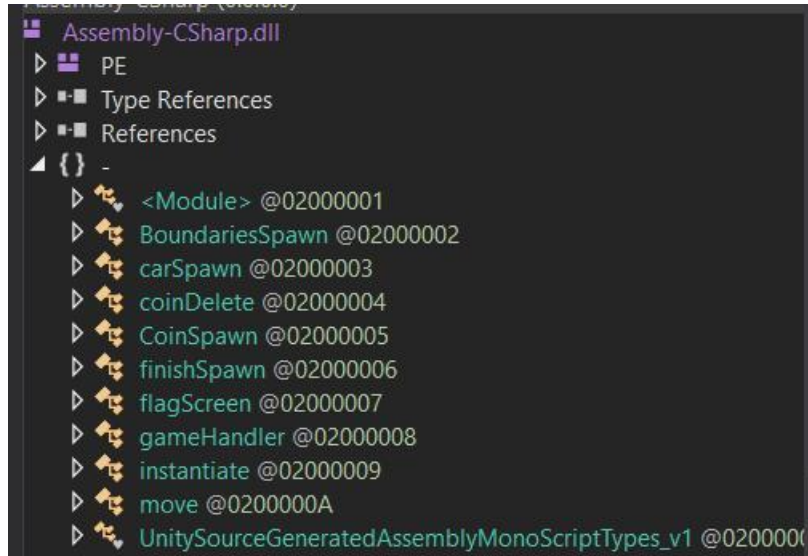
Αυτό κάνει τη ζωή μας πολύ πιο εύκολη, καθώς μπορούμε να χρησιμοποιήσουμε ένα πολύ δυνατό εργαλείο, το DnSpy για να δούμε τον κώδικα του προγράμματος.

Το πλεονέκτημα που έχουμε εδώ είναι ότι το [DnSpy](#):

1. Πραγματοποιεί decompilation πολύ κοντά στον αρχικό κώδικα
2. Επιτρέπει patching του προγράμματος ώστε να μπορούμε να αλλάξουμε τη λειτουργία του.

Φορτώνουμε το αρχείο Assembly-CSharp.dll στο DnSpy μέσω του μενού File → Open...

Κάνουμε expand τα μενού και παίρνουμε την παρακάτω εικόνα:



Εξετάζοντας τα διάφορα scripts, βλέπουμε ότι στο gameHandler υλοποιείται η λειτουργικότητα για τη σύγκρουση του αυτοκινήτου μας (μπλε) με τα μωβ αυτοκίνητα:

```
// Token: 0x0600001C RID: 28 RVA: 0x00002363 File Offset: 0x00000563
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.CompareTag("car"))
    {
        gameHandler.score = 0;
        base.Invoke("Reset", 0f);
    }
}
```

Το παραπάνω μας λέει ότι κάθε φορά που δύο οχήματα συγκρούονται το σκορ μας μηδενίζεται και το παιχνίδι γίνεται reset (το CompareTag χρησιμεύει σαν έλεγχος ότι το αμάξι μας συγκρούστηκε με ένα άλλο αμάξι)


Μπορούμε να κάνουμε το αμάξι μας άτρωτο αφαιρώντας τον έλεγχο του collision (δεξί κλικ → Edit Method...):

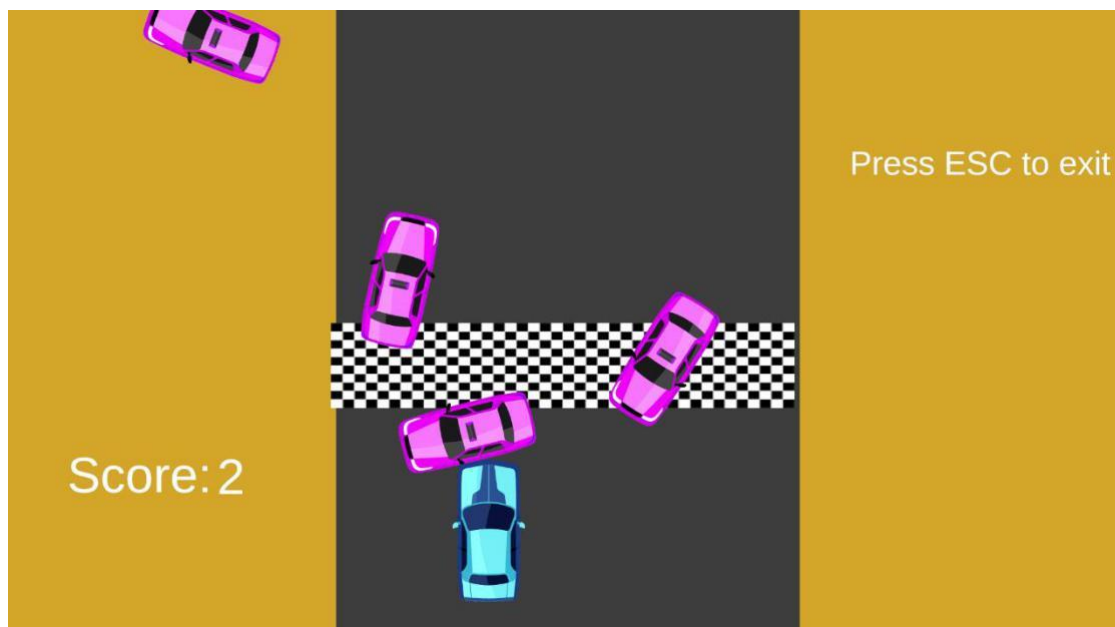
```
5 // Token: 0x02000008 RID: 8
6 public partial class gameHandler : MonoBehaviour
7 {
8     // Token: 0x0600001C RID: 28 RVA: 0x00002363 File Offset: 0x00000563
9     private void OnCollisionEnter2D(Collision2D other)
10    {
11        return;
12    }
13 }
14
```

Τώρα η σύγκρουση δε θα επηρεάζει το όχημά μας.

Εξετάζοντας και άλλο τον κώδικα διαπιστώνουμε την ύπαρξη ενός finish αντικειμένου, το οποίο όταν γίνει trigger (λογικά όταν φτάσει το αμάξι μας σε αυτό), η σκηνή θα αλλάξει στη σκηνή του flag. Ενδιαφέρον:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("coin"))
    {
        gameHandler.score++;
        this.ValueText = GameObject.Find("/Canvas/Score").GetComponent<TextMeshProUGUI>();
        this.ValueText.text = gameHandler.score.ToString();
        return;
    }
    if (other.CompareTag("car"))
    {
        gameHandler.score = 0;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        return;
    }
    if (other.CompareTag("finish"))
    {
        SceneManager.LoadScene("flag");
    }
}
```

Πατάμε αποθήκευση  και ξανατρέχουμε το παιχνίδι:



Πράγματι, υπάρχει finish line, το οποίο διασχίζουμε και...



However, you didn't beat the  
highscore of 10000 points

Μάλιστα, άρα μάλλον πρέπει να πετύχουμε και νέο highscore για να πάρουμε το flag.

Ας επιστρέψουμε στον κώδικα.

Υπάρχει ένα script που λέγεται flagScreen. Ίσως εκεί να υπάρχουν κάποιες πληροφορίες.

Πράγματι, ας δούμε λίγο τη συνάρτηση Update:

```
// Token: 0x06000016 RID: 22 RVA: 0x0000226F File Offset: 0x0000046F
private void Update()
{
    if (this.score >= 10000)
    {
        this.flag.SetActive(true);
    }
    else
    {
        this.nope.enabled = true;
    }
}
```

Η συνάρτηση φαίνεται να ελέγχει αν το τελικό σκορ μας είναι πάνω από 10000 και μόνο τότε ενεργοποιεί το flag. Αλλιώς ενεργοποιείται το κείμενο που αναφέρει την απαίτηση για το highscore. Δεδομένου ότι κάθε νόμισμα μας δίνει 1 πόντο, αυτό είναι αδύνατο να το πετύχουμε κανονικά. Όμως, μπορούμε να αλλάξουμε τον έλεγχο για το score από 10000 σε 0.

```
13 // Token: 0x06000016 RID: 22
14 private void Update()
15 {
16     if (this.score >= 0)
17     {
18         this.flag.SetActive(true);
19     }
```

Τώρα ό,τι και να γίνει, θα περάσουμε τον έλεγχο.

Αποθηκεύουμε και ξανατρέχουμε:



Πράγματι ύστερα από αυτήν την αλλαγή, διασχίζοντας το finish line, πέρα από το τρόπαιο παίρνουμε και το flag!