Digital Whisper Technologic Papers

חלק א' - Cheating is for assholes... And Smarties

מאת יואב שהרבני / Yoav Shaharabani

הקדמה

מאמר זה הוא הראשון בסדרת המאמרים שלי בלהכין צ'יטים למשחק CubeWorld ב-32bit (הורדתי את המשחק בסביבות 2015 אז יכול להיות שדברים השתנו), משחק מסוג ARPG שגרסת האלפא שלו יצאה ב-2013. הצ'יט שנכין יהיה בנוי כ-DLL שנזריק לתהליך המשחק בזמן שהוא רץ.

במאמר זה אנסה להציג את התהליך המלא שעברתי כשהכנתי את הצ'יט, כלומר אציג פה דברים שחשבתי שהם נכונים ומסתבר שטעיתי וגיליתי רק בהמשך, אתקן אותם במאמר הבא.

במאמר זה אציג:

1. מציאת ה-struct של השחקן.
2. מציאת static pointer (פוינטר שהכתובת שלו לא משתנה בריצה מחדש של המשחק).
3. להכין את הצ'יט בהתבסס על שני הדברים שמצאנו.
4. הזרקת DLL פשוטה

קצת על הכלים שנשתמש בהם:

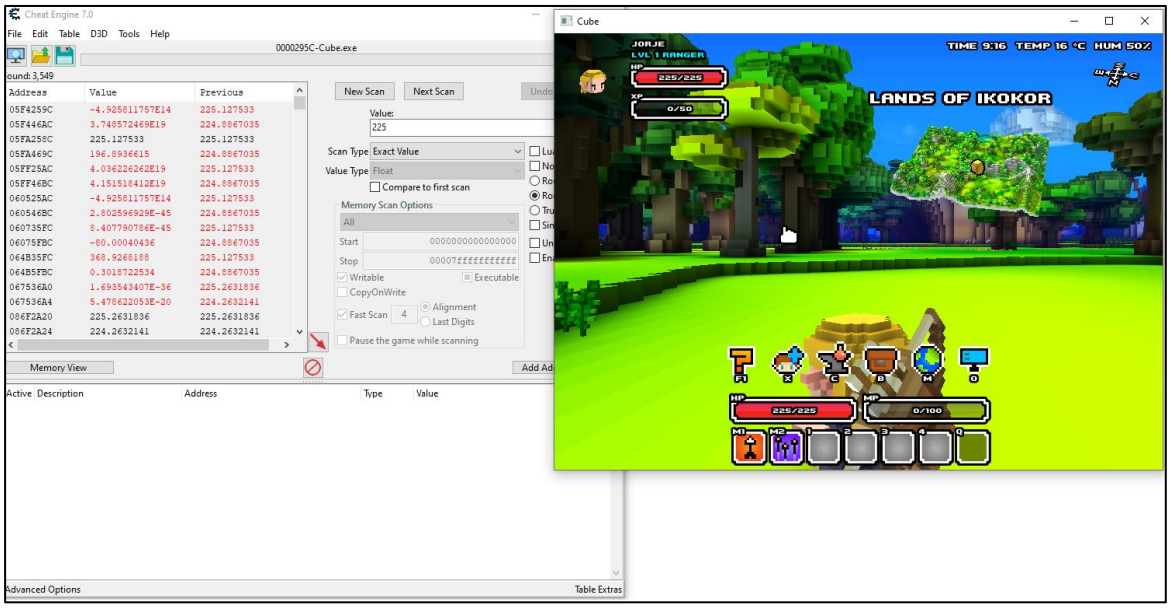
1. Cheat Engine - כלי שנותן לנו את היכולת לסרוק ערכים בזיכרון של תהליך בקלות, נותן לעשות דיבאג על פקודות האסמבלי של התהליך ועוד כל מיני אופציות מועילות שנשתמש בהן. לאלו מכם שאינם מכירים את הכלי או לא מנוסים בו ממליץ להסתכל על המאמר של ניר עופר בנוגע לכלי.
2. ReClass - כלי חזק מאוד שנותן לנו את האפשרות להביט בערכים בזיכרון בתצוגה נוחה כדי לזהות איך ה-structs של המשחק בנויים.
3. Visual Studio 2019, בכלי זה אנחנו נכתוב את ה-DLL שנזריק ונדאבג אותו.

אני מניח שיש לכם ידע בסיסי באסמבלי, בשפת התכנות C או C++, ובזיכרון וירטואלי ומרחבי כתובות.

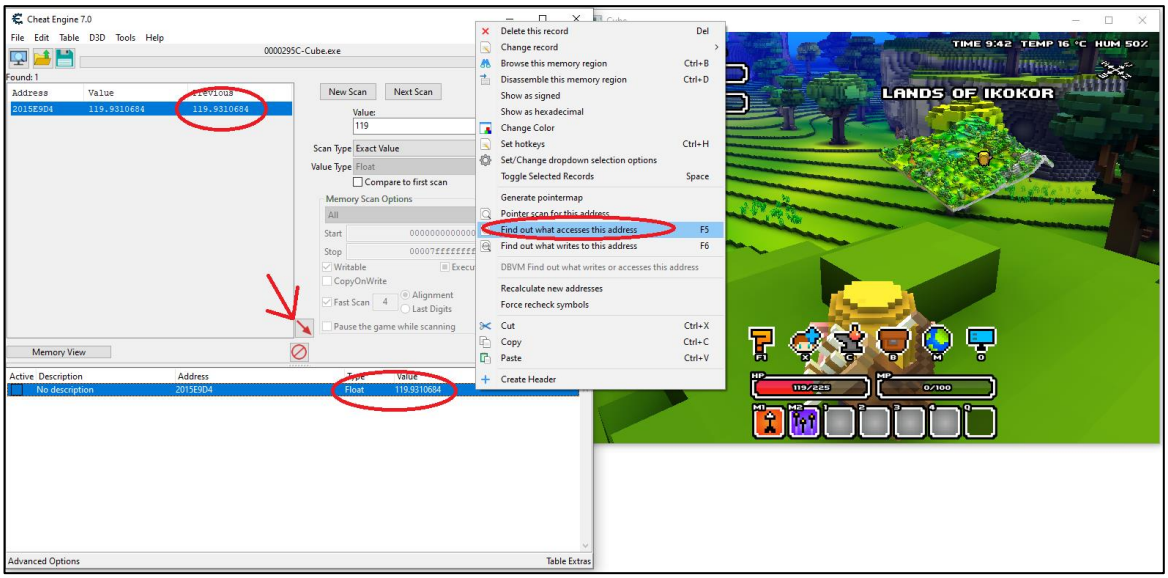
מציאת המבנה של השחקן שלנו

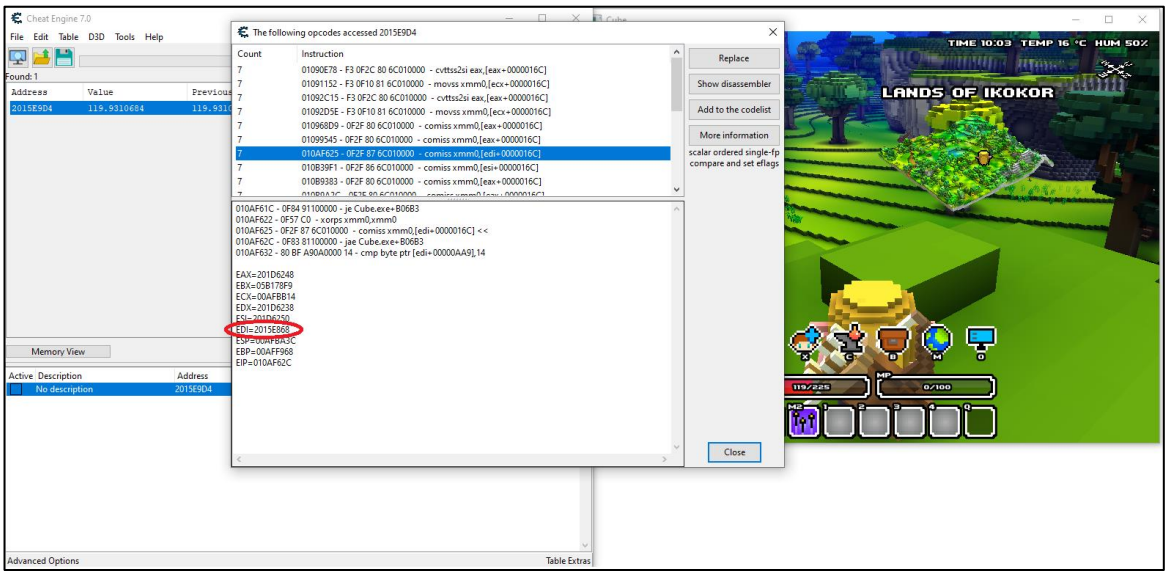
דבר ראשון נפתח את המשחק ונחבר את Cheat Engine אליו.

בהתחלה ננסה למצוא את המבנה שמייצג את כל הנתונים על השחקן שלנו, נתחיל עם Cheat Engine ונחפש את מספר החיים שלנו כערך float, ניפצע ונחפש שוב. בקלות מאוד מצאנו את הערך של החיים שלנו, ננסה לשנות אותו ונראה שהחיים של השחקן שלנו השתנו, כלומר מצאנו את הכתובת הנדרשת.

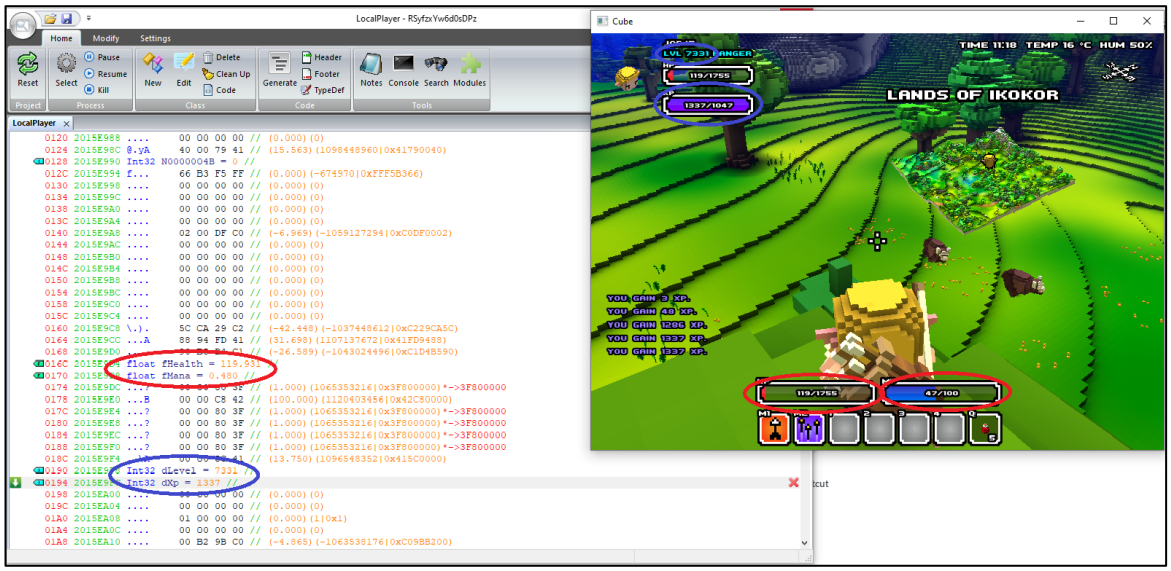


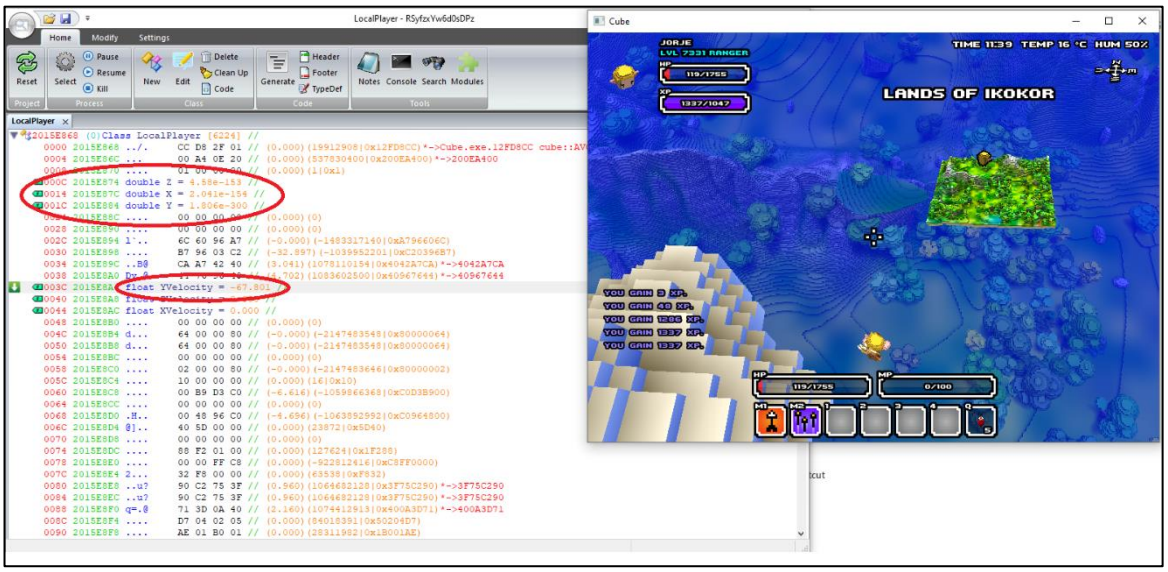
נלחץ מקש ימני על הערך, נקיש על האופציה Find out what accesses this address, זה בעצם יהפוך את Cheat Engine לדיבאגר וכל פעם כשאחת מהוראות האסמבלי גורמות לביצוע פעולה של גישה לכתובת הספציפית הזו, יוצג לנו על המסך הקוד אסמבלי שגרם לביצוע הפעולה ועוד פרטים עליו.





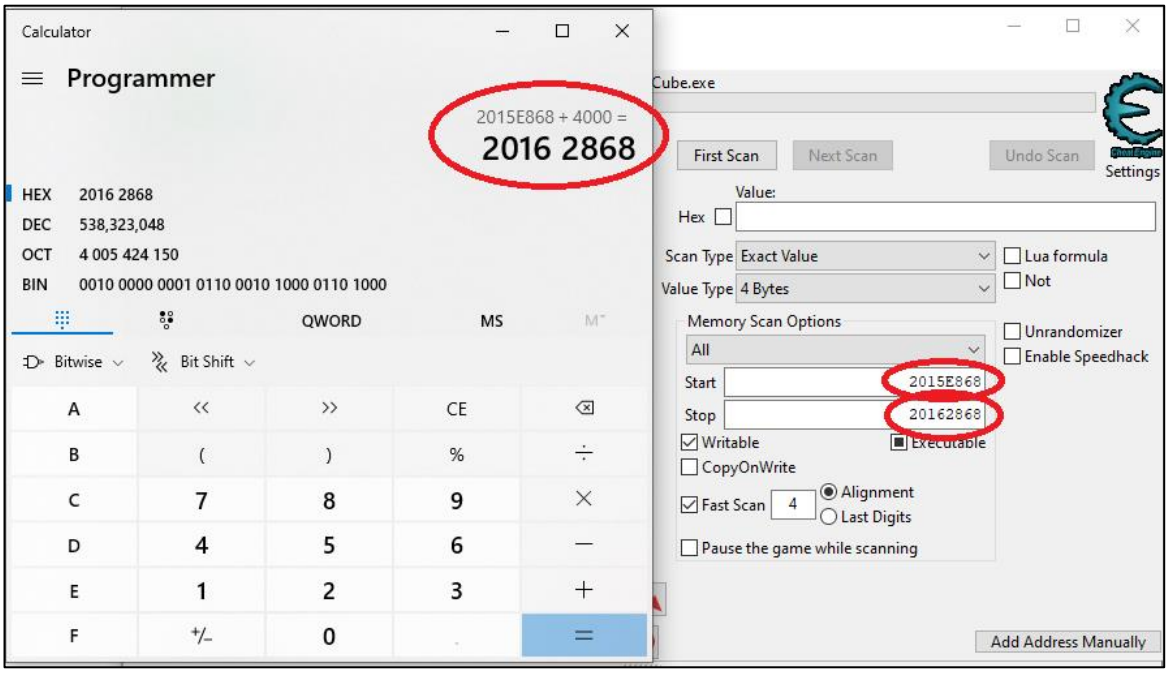
מפה נקבל את ה-Offset C16 מה-struct של השחקן שלנו לערך של החיים. ניקח את הכתובת של ה-struct של השחקן שלנו ונפתח את הכלי ReClass, ניצור מחלקה חדשה ונשים את הכתובת שהעתקנו קודם, כמו שהצגתי למעלה, עם כלי זה נוכל למצוא את ה-struct של השחקן בקלות. נרד למטה ל- Offset 16C ונסמן את הנתונים שם כ-float ונקרא להם fHealth. בקלות נוכל לשחק עם כמות המנה שיש לנו ולגלות שה-Offset של המנה הוא 170 וגם הוא float. לאחר מכן ננסה לזוז טיפה במפה ולפי זה נגלה לא רק את הקאורדינטות אלא גם את הערכים שמסמלים את המהירות שלנו לכיוון מסויים, כך למשל, באיור 2 שיניתי את המהירות שלנו בציר ה-Y ל-100 וקיבלנו "SuperJump". לאחר פעולות אלו מצאנו חלק קטן אך חשוב ומספק מהמבנה של השחקן.





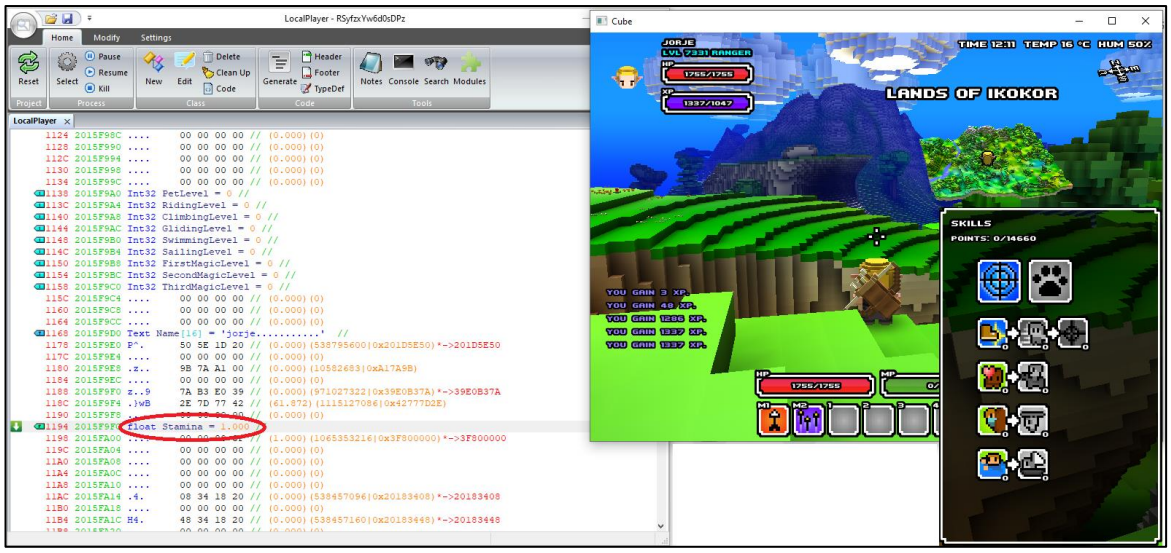
נשים לב גם שתמיד הבייט הראשון יהיה CC כלומר 204 בדצימלי, נשמור פרט זה שכן הוא יעזור לנו מאוחר יותר (ReClass הציג את ארבעת בייטים הראשונים כמשתנה ידוע של המשחק בשם AVCreature, זה תחום שיותר נוגע ל-vtables ואגע בו במאמר הבא).

בנוסף קיים פיצ'ר ב-Cheat Engine שיעזור לנו גם לחפש דברים, למשל בעזרתו מצאתי את ה-Stamina של השחקן שלנו.



כאן ניתן לראות שבחרנו את הטווח הכתובות שבהן נחפש את הערכים, בגלל שאנחנו מניחים שהערך של ה-Stamina יימצא איפשהו בתוך ה-struct של השחקן שלנו, נשים את הכתובת של ה-struct בתור Starting Address, ובתור Ending Address נשים את הכתובת ועוד 4000(ב-hex, ניחוש קל שגודל ה-struct יהיה קטן מזה) ומכאן הסריקות יהיו מצומצמות ומהירות יותר מהסריקות על כל הכתובות.

נוכל בעזרת שיטה זו למצוא גם את ה-Stamina בקלות וגם כל מיני ערכים שמייצגים את ה SkillLevel בכל מיני מיומנויות.



לבסוף נלחץ על הכפתור Generate ונקבל class יפה שמייצג את השחקן שלנו שמכיל גם Paddings של המשתנים שלא הגדרנו.

Class Code Generated

File

```
1 // Generated using ReClass 2016
2
3 class LocalPlayer;
4
5 class LocalPlayer
6 {
7 public:
8     char pad_0x0000[0xC]; //0x0000
9     double Z; //0x000C
10    double X; //0x0014
11    double Y; //0x001C
12    char pad_0x0024[0x18]; //0x0024
13    float YVelocity; //0x003C
14    float ZVelocity; //0x0040
15    float XVelocity; //0x0044
16    char pad_0x0048[0xE0]; //0x0048
17    __int32 N0000004B; //0x0128
18    char pad_0x012C[0x40]; //0x012C
19    float fHealth; //0x016C
20    float fMana; //0x0170
21    char pad_0x0174[0x1C]; //0x0174
22    __int32 dLevel; //0x0190
23    __int32 dXp; //0x0194
24    char pad_0x0198[0xFA0]; //0x0198
25    __int32 PetLevel; //0x1138
26    __int32 RidingLevel; //0x113C
27    __int32 ClimbingLevel; //0x1140
28    __int32 GlidingLevel; //0x1144
29    __int32 SwimmingLevel; //0x1148
30    __int32 SailingLevel; //0x114C
31    __int32 FirstMagicLevel; //0x1150
32    __int32 SecondMagicLevel; //0x1154
33    __int32 ThirdMagicLevel; //0x1158
34    char pad_0x115C[0xC]; //0x115C
35    char Name[16]; //0x1075888
36    char pad_0x1178[0x1C]; //0x1178
37    float Stamina; //0x1194
38    char pad_0x1198[0x16C]; //0x1198
39    __int32 Copper; //0x1304
40    char pad_0x1308[0x548]; //0x1308
41
42 }; //Size=0x1850
43
```

עוד משהו קטן שנעשה דרך CheatEngine אנחנו נרצה שנוכל לבטל כל Damage שנקבל כתוצאה מנפילה, אז נלחץ על הערך עם מקש ימני ונלחץ Find out what writes to this address, אופצייה זו תיתן לנו רשימה של כל דבר שמשנה את הערך.

עכשיו ניפול בכוונה ממקום גבוה ונראה שקפצה פקודת אסמבלי חדשה שלא ראינו קודם, ניכנס אליה נלחץ מקש ימני, Replace with code that does nothing, וזה ישנה את שורת האסמבלי הזו ל-NOP, או בבייטים 0x90 על כל בייט בשורה (8 סך הכל) ולאחר השינוי בכל פעם שניפול לא נקבל Fall Damage.

Cube.exe+21C323	F3 0F11 80 6C010000	movss	[eax+0000016C],xmm0

Cube.exe+21C323	90	nop
Cube.exe+21C324	90	nop
Cube.exe+21C325	90	nop
Cube.exe+21C326	90	nop
Cube.exe+21C327	90	nop
Cube.exe+21C328	90	nop
Cube.exe+21C329	90	nop
Cube.exe+21C32A	90	nop

נוסיף את זה כעוד פיצ'ר ב-DLL שנכין.

מציאת מצביע סטטי לשחקן שלנו.

טיפה על מצביעים סטטיים ואיך הם נראים בקוד

מצביעים סטטיים הם בעצם משתנים בקוד שהוגדרו כמשתנים סטטיים שמכילים בתוכם כתובת לאובייקט הבא בזיכרון, למשל קיים ה-World struct, ובתוכו קיים מצביע ל-Map struct, ובתוך Map קיים מצביע ל-Player struct שהוא מייצג את השחקן שלנו, ונניח שיש לנו משתנה סטטי שנקרא MyWorld מסוג מצביע ל-World, בעצם גישה לחיים של השחקן תיהיה זה יראה כך:

MyWorld->MyMap->MyPlayer->fHealth

כלומר בקוד האסמבלי זה יראה כך:

```
Mov eax, [MyWorld]
Mov eax, [eax + Offset_To_MyMap]
Mov eax, [eax + Offset_To_MyPlayer]
Mov eax, [eax + Offset_To_fHealth]
```

לאחר הפקודה הראשונה נשיג את הכתובת ל-struct מסוג World הרלוונטי עבורנו, לאחר הפקודה השנייה נשיג את הכתובת ל-MyMap וכן הלאה עד שנגיע לכתובת של החיים ונכניס ל-eax את כמות החיים שלנו.

הסבר קטן על מהו ASLR ולמה הוא קשור ל-Static Pointers.

ASLR או Address Space Layout Randomization היא טכניקה שנועדה למנוע ניצול של חולשות בזיכרון, היא עושה את זה בכך שהיא מסדרת באופן רנדומלי את מרחב כתובות הזיכרון, אני לא אסביר למה היא עובדת (ואיך עדיין עוקפים אותה ☺) אבל אם מעניין אתכם תוכלו להמשיך לחקור על זה.

לדוגמא, נניח שלקובץ ההרצה שלנו קיים המשתנה הסטטי MyWorld, ונניח שהרצנו את הקובץ – כלומר הקובץ הועלה לזיכרון והתחיל לרוץ.

בזמן הריצה ראינו שהמשתנה הסטטי שלנו קיבל את הכתובת 0xCD500. עכשיו נצא מהמשחק ונפתח מחדש, עכשיו נראה כי הכתובת של המשתנה הסטטי השתנתה והפעם היא 0xDE500.

מה בעצם קרה כאן? למה משתנה סטטי שהכתובת שלו לא אמורה להשתנות משתנה, ואיך אנחנו עדיין יכולים להשתמש בו?

כשהעלנו את המשחק לזיכרון (נניח המשחק הוא Cube.exe) בפעם הראשונה, ה-BaseAddress שלו הועלה ל-0xCD000, לעומת זאת בפעם השנייה שבה העלנו את המשחק לזיכרון, ה-BaseAddress שונה ל-0xDE000. בשני המקרים הכתובת של המשתנה הסטטי MyWorld פחות ה-BaseAddress היא 0x500, כלומר אם נשיג את ה-BaseAddress של המשחק(שנוכל להשיג אותו בקלות) נוכל להוסיף לו 0x500 ולהגיע למשתנה הסטטי MyWorld.

טיפ קטן: אפשר לומר ל-CheatEngine לגשת אל המשתנה הזה בקלות אם נשים "Cube.exe" + 500.

כמובן שהמשתנים הסטטיים שלנו יכולים להיות מוגדרים בתוך אחד ה-DLL-ים של המשחק, ובמקרה כזה נאלץ למצוא את ה-BaseAddress של ה-DLL הספציפי שהוא מוגדר בתוכו ומשם לגשת אליו.

לכל DLL או exe שהועלה לזיכרון של התהליך נקרא "Module" ולכל Module כזה יהיה BaseAddress וגם משתנים ופקודות אסמבלי שנמצאות במרחק קבוע ממנו.

Pointer Scan

סריקת מצביעים היא אופציה ב-CheatEngine שבעצם התוכנה סורקת את כל מרחב הכתובות של כל Module שבתוך התהליך. לכל Module קיים BaseAddress ו-SizeOfModule ועל ידי שילוב של השניים ניתן לעבור על הכתובות.

על כל כתובת ש-CheatEngine חושד שמכילה ערך שהוא כתובת אחרת, כלומר כל כתובת שהיא בעצם מצביע לאנשהו, CheatEngine ימשיך איתה ויבדוק לאן היא מצביעה או אם קיים אליה ערך קרוב שגם הוא מצביע.

קל לראות שזו בעיית גרף קלאסית, נעבור על כל הכתובות ונחפש את המסלול שמגיע מכתובת סטטית לכתובת שנרצה (מסלול באורך מקסימלי N), כשבכל פעם שנעשה קפיצה מכתובת, ננסה להוסיף לה offset-ים שונים.

נריץ את זה עם ההגדרות הדיפולטיביות (N=7) ונחפש מסלול שמגיע לכתובת של השחקן שלנו, ונקבל 727,85661, מסלולים, מספר דיי גדול שנצמצם אותו לאט לאט.

Pointer scan : FirstScan.PTR

File Distributed pointer scan Pointer scanner

4 Bytes — Pointer paths:61727856

Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5	Offset 6	Points to:
"Cube.exe"+0028515C	71C	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+00285150	7C8	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+0009C810	390	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+0009A750	850	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"MSVCP110.dll"+0001ECF8	A98	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"MSVCP110.dll"+00024E...	A98	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+0037F950	9DC	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+0018BD7C	B14	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"0Kraken0510DevProps.d...	E64	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"MSVCP110.dll"+0006F2...	E68	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"Cube.exe"+0037ED14	774	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"MSVCR110.dll"+000D1B24	7D0	850	3B4	6C0	2C	0	0	2015E868 = 19912908
"MSVCR110.dll"+000A94...	B7C	850	3B4	6C0	2C	0	0	2015E868 = 19912908

אפשר לראות בצד שמאל של התמונה שכמו שהסברתי קודם כל הכתובות הן מהצורה ModuleName + RVA (למי שלא מכיר RVA היא כתובת יחסית).

אפשר לראות גם בכל אחד מהמסלולים את כל ה-offset-ים שלקחו חלק במסלול.

נצא מהמשחק ונריץ אותו מחדש, נעשה את כל התהליך של מציאת הכתובת למבנה של השחקן ונריץ Rescan והפעם נחפש את הכתובת החדשה שלנו, רוב המסלולים לאחר פתיחה מחדש של המשחק יהיו כבר לא רלוונטים עבורנו, שכן הם כבר לא מצביעים על הכתובת של המבנה של השחקן שלנו, ולכן CheatEngine יסיר אותם ויצמצם לנו את כמות התוצאות.

צריך לשים לב שלאחר פתיחה מחדש של המשחק צריך לעשות Attach למשחק מחדש בצד שמאל למעלה ב-CheatEngine. אחרי הריצה השנייה נקבל 13 מסלולים.

Pointer scan : SecondScan.PTR

File Distributed pointer scan Pointer scanner

4 Bytes — Pointer paths:13

Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5	Offset 6	Points to:
"Cube.exe"+0036B1CC	34	18	AC	3F4	0	0		1FD24328 = 19912908
"Cube.exe"+0036B1CC	154	0	24	AC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	154	4	24	AC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	154	24	AC	3F4	0	0		1FD24328 = 19912908
"Cube.exe"+0036B1CC	70	24	124	AC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	84	38	4	AC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	34	18	AC	4D8	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	154	24	AC	4D8	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	34	18	AC	4FC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1CC	154	24	AC	4FC	3F4	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1BC	8	34	0	74	2C	0	0	1FD24328 = 19912908
"Cube.exe"+0036B1BC	8	34	0	74	20	0		1FD24328 = 19912908
"Cube.exe"+0036B1BC	C	70	34	0	74	20	0	1FD24328 = 19912908

נריץ מחדש שוב את המשחק ונלחץ פעמיים על הכתובות שעדיין שוות לכתובת לשחקן שלנו, נקבל אותן במצג נוח למטה:

Active	Description	Address	Type	Value
	No description	20B3D078	Float	0
	pointerscan result	P->20B21078	4 Bytes	19912908
	pointerscan result	P->????????	4 Bytes	??
	pointerscan result	P->????????	4 Bytes	??
	pointerscan result	P->????????	4 Bytes	??
	pointerscan result	P->20B21078	4 Bytes	19912908
	pointerscan result	P->????????	4 Bytes	??
	pointerscan result	P->????????	4 Bytes	??
	pointerscan result	P->20B21078	4 Bytes	19912908

נשים לב למשהו מוזר, הכתובות הסטטיות האלו כן מגיעות ל-struct של השחקן שלנו, אבל משום מה הן משתנות מידי פעם לערכים לא ידועיים.

הערה: במאמר הבא אחרי Reverse ב-Ghidra אני אמצא איך האובייקט של העולם בנוי וממנו אגיע אל המבנה של השחקן שלנו בצורה נוחה קלה ואלגנטית.

בקוד שנכתוב נצטרך להתייחס לבעיה הזו. נבחר את המסלול עם ה-offset-ים האלו:

```
const size_t offsetsLocalPlayer[] = { 0x36B1CC, 0x34, 0x18, 0xAC, 0x4FC, 0x3F4, 0 };
```

כתיבת הצ'יט

נפתח את VisualStudio וניצור פרוייקט חדש של Dynamic Link Library, וכמובן שבתור שלב 0 נעשה בתחילת הקובץ:

```
#include <Windows.h>
```

נתחיל מיצירת קבצי העזר שלנו:

Cube_structs.h:

קובץ header זה יהיה אחראי על כל ה-struct-ים שמצאנו על ידי ReClass, פשוט נעתיק את מה שהכלי יצר אחרי שלחצנו Generate ונשים בקובץ.

הקומפיילר עלול להוסיף paddings למחלקות או למבנים שאנחנו יוצרים, בגלל שאנחנו רוצים שהמחלקה שבנינו תיהיה תואמת בדיוק מוחלט לאובייקט בזיכרון ניצטרך להוסיף בתחילת הקובץ את הפקודה:

```
#pragma pack(1)
```

פקודה זו אומרת לקומפיילר שינסה להוסיף padding כדי להשלים ל-1, כלומר שלא יוסיף padding (אלא אם אנחנו מתעסקים עם ביטים ולא עם בייטים ואז הוא ישלים לבייט אחד).

נניח היינו שמים את זה עם הארגומנט 4, במידה וניצור struct או class בצורה כזו:

```
Struct Player{
        char Name[3];
        int age;
}
```

נגלה שאם נעשה sizeof(Player) התוצאה תיהיה 8 במקום 7, משמע הקומפיילר הוסיף padding בין ה-int למערך ה-char-ים.

Memory.cpp:

בקובץ זה יהיו לנו פונקציות שאחראיות על כל מה שקשור לזיכרון של התהליך והמניפולציות שנעשה בו. אנחנו מניחים בפונקציות אלו שהמרחב הוירטואלי של ה-DLL ושל התהליך הוא אותו מרחב וירטואלי כי בעצם אנחנו נזריק את ה-DLL לתהליך.

נתחיל מהפוקנצייה:

```
DWORD GetModuleBase(LPCWSTR moduleName)
```

פונקצייה זו בעצם תקבל שם של Module (למשל "Cube.exe") ותחזיר את ה-BaseAddress שלו. היא תיראה כך:

```
DWORD GetModuleBase(LPCWSTR moduleName)
{
    DWORD hModule = (DWORD)GetModuleHandle(moduleName);
    return hModule;
}
```

נעבור לפונקצייה הבאה, GetDynamicAddress, פונקצייה זו תקבל את ה-BaseAddress, את ה-offset-ים של המסלול שמצאנו בסריקת המצביעים שעשינו ואת מספר ה-offset-ים ותחזיר לנו את הכתובת של המבנה שמצאנו. במידה ואחד מהמצביעים במהלך המסלולים היו לא חוקיים, אז זה יחזיר 0 במקום שהתוכנה תקרוס. היא תיראה כך:

```
DWORD GetDynamicAddress(DWORD ModuleBase, const size_t offsets[], size_t numOffsets)
{
    uintptr_t bDynamicAddress = ModuleBase;

    for (size_t i = 0; i < numOffsets; i++)
    {
        bDynamicAddress += offsets[i];
        if (!IsBadReadPtr((LPVOID)bDynamicAddress, sizeof(int)))
        {
            bDynamicAddress = *(uintptr_t*)bDynamicAddress;
            if (!bDynamicAddress)
            {
                return 0;
            }
        }
        else
        {
            return 0;
        }
    }
    return bDynamicAddress;

}
```

נעבור לפונקצייה PatchCode, שתקבל את הכתובת של הקוד את מספר הבייטים שנחליף ומערך של בייטים שנחליף אליהם והיא בעצם תחליף את הקוד אסמבלי וקוד אסמבלי אחר לבחירתינו, לבסוף היא תחזיר את קוד האסמבלי הישן שהחלפנו. והיא תיראה כך:

```
BYTE* PatchCode(LPVOID dAddressCode, BYTE bCode[], size_t szCodeSize)
{
    DWORD dOldProtect;
    BYTE* oldCode;

    if (!bCode)
    {
        return NULL;
    }

    oldCode = (BYTE*)malloc(sizeof(BYTE) * szCodeSize);
    if (oldCode)
    {
        VirtualProtect(dAddressCode, szCodeSize, PAGE_EXECUTE_READWRITE, &dOldProtect);
        memcpy(oldCode, dAddressCode, szCodeSize);
        memcpy(dAddressCode, bCode, szCodeSize);
        VirtualProtect(dAddressCode, szCodeSize, dOldProtect, &dOldProtect);
    }

    return oldCode;
}
```

בעצם בגלל שההרשאות על איזור הקוד הן רק READ_EXECUTE, לא נוכל פשוט לרשום שם מה שנרצה, אחרת המשחק יקרוס עם ACCESS VIOLATION, ולכן נשתמש בפונקציית ה-API של Windows שנקראת VirtualProtect ונחליף את ההרשאות על איזור זה לPAGE_EXECUTE_READWRITE ואז נהייה רשאים לכתוב שם, ואז נשנה את ההגדרות בחזרה למה שהיה. הפונצייה האחרונה תיהיה NopCode, פונקצייה זו פשוט תשתמש בפונקצייה הקודמת שיצרנו והיא פשוט תחליף את הבייטים ל-0x90, כלומר תהפוך את כולם ל-NOP באסמבלי.

```
BYTE* NopCode(LPVOID dAddressCode, size_t szCodeSize)
{
    BYTE* newCode;
    BYTE* bCode = (BYTE*)malloc(sizeof(BYTE) * szCodeSize);
    if (bCode)
    {
        memset(bCode, 0x90, szCodeSize * sizeof(BYTE));
        newCode = PatchCode(dAddressCode, bCode, szCodeSize);
        free(bCode);
    }
    return newCode;
}
```

:Dllmain.cpp

פה תיהיה פונקצייה ה-main של ה-DLL שלנו, בתור שלב 0 ניצור פונקצייה חדשה שהיא תהווה פונקצייה ה-Thread שלנו ובה הצ'יט יתנהל, היא תיהיה מהצורה הזו:

```
DWORD WINAPI CheatThread(LPVOID lParam)
```

ונרצה ליצור Thread חדש שירוץ על הפונקצייה הזו ונשלח לו כפרמטר את ה-hModule כדי שנוכל מאוחר יותר לומר למערכת ההפעלה לשחרר אותו.

```
BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)CheatThread, hModule, 0, 0);
        if (hThread)
        {
            CloseHandle(hThread);
        }

        break;
    }
    return TRUE;
}
```

נרצה לעשות את זה כי התהליך מחליט אם ה-DLL הוזרק (או במקרה אחר הועלה) לזיכרון בצורה טובה לפי הערך המוחזר מהפונקצייה ולא נרצה שיחכה לו עד שנצא מהתהליך.

עכשיו נתחיל לעבוד על הפונקצייה CheatThread: דבר ראשון נרצה שיהיה לנו shell אינטראקטיבי כשנזריק את ה-DLL לתהליך. נגדיר משתנה גלובאלי שנקרא HANDLE hConsole שממנו נכתוב ל-shell שניצור. נשתמש בפונקצייה ;AllocConsole() כדי ליצור את ה-Console שלנו ואז נשתמש בפונקצייה GetStdHandle(STD_OUTPUT_HANDLE) ונשים את הערך המוחזר ב-hConsole.

בסוף הפונקצייה נשתמש ב-;FreeConsole() ואז ב FreeLibraryAndExitThread() כדי להודיע למערכת ההפעלה שסיימנו עם ה-DLL והיא יכולה למחוק אותו מהזיכרון (אני לא אוסיף יותר פרטים לגבי זה, אם תרצו תוכלו לקרוא על הפונקצייה הזו בדוקומנטציה של Windows API). היא תקבל את lParam בתור HMODULE.

ניצור פונקצייה חדשה שהיא תיהיה אחראית על כל נושא הכתיבת ל-Console שתיראה כך:

```
VOID DllPrint(LPCWSTR text)
{
    WriteConsole(hConsole, text, wcslen(text), 0, NULL);
}
```

כעת נגדיר כמה משתנים קבועים כמו השם של ה-Module, את ה-offset-ים שמצאנו, ומשתנה בוליאני ששווה ל-FALSE שמייצג לנו האם מצאנו את ה-Address של ה-struct של השחקן שלנו, בנוסף נגדיר את המשתנה dAVCreature ששווה ל-204, שזה בעצם מה שמצאנו קודם שזה תמיד הבייט הראשון במבנה של השחקן.

נשתמש ב-GetModuleBase כדי למצוא את ה-BaseAddress של "Cube.exe". ואז ניכנס ללולאה עד שנמצא את הכתובת של השחקן שלנו. היא תיראה כך:

```
while (!foundDynamicAddress)
{
    Sleep(100);
    dDynamicAddressLocalPlayer = GetDynamicAddress(dModuleBaseAddress, offsetsLocalPlayer,
        int(sizeof(offsetsLocalPlayer) / sizeof(size_t)));

    /* Checks if the returned value is a valid pointer. */
    if (!IsBadReadPtr((LPVOID)dDynamicAddressLocalPlayer, sizeof(int)))
    {
        /* Checks if the first BYTE is 0xCC in hex or 204 in decimal. */
        if (*(BYTE*)dDynamicAddressLocalPlayer == dAVCreature)
        {
            foundDynamicAddress = TRUE;
        }
    }
}
```

לאחר מכן נדפיס ל-Console שלנו שמצאנו את הכתובת של השחקן. ונגדיר:

```
PLocalPlayer pLocalPlayer;
pLocalPlayer = (PLocalPlayer)(dDynamicAddressLocalPlayer);
```

הערה:

- הגדרתי ש-PLocalPlayer יהיה בעצם מצביע ל-LocalPlayer.
- אני לא אסביר על כל הפיצ'רים של הצ'יט כי הבנייה של רוב הפיצ'רים חוזרת על עצמה, אני אקח רק שלושה פיצ'רים בתקווה שיציגו בצורה הטובה ביותר את הצ'יט.

כעת ניכנס ללולאה של הצ'יט, בסוף כל חזרה של הלולאה יחכה לה Sleep(30) (זה יהיה כביכול ה-"GameTick" שלנו).

קודם כל נתחיל מאינטראקציה עם המשתמש, כדי לקבל קלט מהמקלדת נשתמש בפונקצייה GetAsyncKeyState, נתחיל מאם המשתמש לוחץ על VK_NUMPAD0, נצא מהלולאה ונסיים את ה-Thread.

```
while (!bStopped)
{
    /* Stop */
    if (GetAsyncKeyState(VK_NUMPAD0) & 1) {
        bStopped = TRUE;
    }
```

נמשיך במקש VK_NUMPAD1, שאם המשתמש לוחץ עליו החיים שלו נעצרים, לא עולים ולא יורדים. נגדיר שני משתנים מחוץ ללולאה, הראשון BOOL bHaltHealth, השני float fLastHealth, ברגע שנלחץ על VK_NUMPAD1 נשנה את bHaltHealth להופכי של עצמו ונשמור את מספר החיים שלנו ב fLastHealth. כמובן שנרצה גם להדפיס הודעה רלוונטית ל-Console:

```
/* HaltHealth */
if (GetAsyncKeyState(VK_NUMPAD1) & 1) {
    fLastHealth = pLocalPlayer->fHealth;
    bHaltHealth = !bHaltHealth;
    DllPrint(L"VK_NUMPAD1: TOGGLE HALT HEALTH\n");
}
```

בהמשך הלולאה נבדוק את הערך של המשתנה bHaltHealth ונפעל בהתאם:

```
if (bHaltHealth)
{
    pLocalPlayer->fHealth = fLastHealth;
}
```

בצורה מאוד דומה נוכל לעשות את זה גם על ה-Stamina ועל ה-Mana שלנו, ונוכל להוסיף בצורה מאוד פשוטה גם Copper (פשוט נוסיף לערך מספר כלשהו, למשל 100).

נמשיך ונרצה ליצור FlyHack, גם פה כשלוחצים על המקש R נשים את הערך TRUE במשתנה bStartFlyHack, ולאחר מכן בסוף הלולאה נבדוק מה מצב המשתנה ולפיו נפעל בהתאם.

זוכרים את הערך שמסמל את המהירות שמצאנו במבנה של השחקן קודם? כאן נשתמש במהירות לפי ציר ה-Y. במידה ואנחנו במצב ה-FlyHack שלנו, אם אנחנו לוחצים על מקש רווח נרצה לעלות כלומר נציב במהירות כלפי מעלה 10, כמובן שככל שנשים מספר גבוה יותר ככה נעלה יותר מהר כלפי מעלה. לעומת זאת אם נלחץ על מקש ה-LeftControl נשים מהירות כלפי מטה של 10.

לבסוף אם לא לחצנו על אף אחד מהמקשים האלו, אנחנו נרצה לרחף למרות "כוח הכבידה" שקיים במשחק, ולכן נציב את המהירות בציר ה-Y כ-0.

```
if (bStartFlyHack)
{
    if (GetAsyncKeyState(VK_SPACE) & 1)
    {
        /* Give 10 velocity up. */
        pLocalPlayer->YVelocity = 10;
    }
    else if (GetAsyncKeyState(VK_LCONTROL) & 1)
    {
        /* Give 10 velocity down. */
        pLocalPlayer->YVelocity = -10;
    }
    else
    {
        /* Halt player YVelocity */
        pLocalPlayer->YVelocity = 0;
    }
}
```

הדבר האחרון שאראה לכם הוא איך נבטל את ה-FallDamage.

דבר ראשון ניקח את הכתובת היחסית שמצאנו לפקודת האמבלי שמורידה את החיים שלנו בנפילה ונוסיף ל-BaseAddress שלנו.

שמתי את הצ'יט הזה על המקש VK_NUMPAD5, כשנלחץ עליו אם ה-FallDamage פועל נרצה לבטל אותו, אז נשתמש בפונקציית ה-NopCode שהכנו קודם ונשים אותה על 8 בייטים ועל הכתובת שמצאנו לפקודה. כלומר עכשיו במקום שתוריד לנו חיים בנפילה, היא תעשה כלום.

נצטרך לשמור את ערך החזרה מהפונקצייה NopCode שזה בעצם הבייטים הישנים, כדי שבמקרה שנרצה לבטל את הצ'יט נוכל להחזיר אותם.

כמובן גם שנדפיס הודעה מתאימה ל-Console.

```
/* Stop fall damage. */
if (GetAsyncKeyState(VK_NUMPAD5) & 1) {
    if (!bDisableFallDmg)
    {
        bLastCodeFallDmg = NopCode((LPVOID)dAddressFallDmg, 8);
        bDisableFallDmg = TRUE;
        DllPrint(L"VK_NUMPAD5: Fall dmg off. \n");
    }
    else
    {
        if (bLastCodeFallDmg)
        {
            PatchCode((LPVOID)dAddressFallDmg, bLastCodeFallDmg, 8);
            free(bLastCodeFallDmg);
            bDisableFallDmg = FALSE;
            DllPrint(L"VK_NUMPAD5: Fall dmg on. \n");
        }
    }

}
```

לחדים מבינכם ששואלים את עצמם למה בעצם לא השתמשנו ב-NopCode בשביל למנוע מהחיים שלנו לרדת כשיורים בנו זה כי כל הדמויות במשחק יורשות מאותה מחלקה AVCreature את הפונקציות שלה, כולל זאת שמורידה להם חיים במידה וירו בהם, כלומר אם נשנה את זה ל-NOP, אומנם לנו לא ירדו חיים אבל גם לשאר ה-NPC-ים לא ירדו החיים כשנירה בהם, תוצאה שלא נרצה להגיע אליה.

באותה צורה גם אם נעשה NOP לפונקצייה שמגדירה את התאוצה כלפי מטה ("כוח הכבידה") אז כל הדמויות ירחפו וגם בזה אנחנו לא רוצים.

DLL Injection

עד כה דיברנו על הכנת ה-Payload - ה-DLL שאותו נזריק ובו הלוגיקה של הצ'יט. בחלק זה אציג דרך מאוד פשוטה לביצוע של הזרקת DLL לתוך תהליך רץ על מנת לגרום ל-DLL להשפיע על התהליך.

ניצור פרוייקט חדש ב-Visual Studio (Console Application), נשתמש בספריות "Windows.h" ו-"TLHelp32.h". נתחיל מלכתוב פונקצייה שתמיר לנו שם של תהליך ל-PID שלו כדי שנוכל לעבוד איתו:

```
DWORD GetProcId(LPCWSTR procName)
{
    DWORD procId = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (hSnap != INVALID_HANDLE_VALUE)
    {
        PROCESSENTRY32 procEntry;
        procEntry.dwSize = sizeof(procEntry);

        if (Process32First(hSnap, &procEntry))
        {
            do
            {
                if (!_wcsicmp(procEntry.szExeFile, procName))
                {
                    procId = procEntry.th32ProcessID;
                    break;
                }
            } while (Process32Next(hSnap, &procEntry));
        }
    }
    CloseHandle(hSnap);
    return procId;
}
```

אני לא אכנס יותר מידי לאיך הפונקצייה עובדת אבל מי שמעוניין מוזמן להסתכל בדוקומנטציה של Microsoft על הפונקצייה CreateToolhel32Snapshot עם ה-TH32CS_SNAPPROCESS Flag.

כעת, נעבור ל-main שלנו, נתחיל מלהגדיר כמה משתנים:

```
LPCWSTR dllPath = L"C:\\Path\\To\\Dll.dll";
LPCWSTR procName = L"Cube.exe";
DWORD procId = 0;
```

הערה:

- כשאני מפתח ב-Windows אני משתדל כמה שיותר להשתמש במשתנים מ-Windows API, כמובן שלא חייב אבל קיימת דוקומנטציה נרחבה לגבי הסוגים האלה.
- אני מוסיף L לפני ה-strings מפני שאני משתמש ב-WideCharacters, מציע לכם לקרוא על זה בדוקומנטציה.

בשלב זה נחכה עד שנשיג את ה-PID של התהליך (במידה והתהליך לא עלה עדיין נחכה לו):

```
while (!procId)
{
    procId = GetProcId(procName);
    Sleep(30);
}
```

כאן מתחיל הקטע המעניין, כמו שרשום בהערות, קודם נקבל HANDLE לתהליך על ידי הפונקצייה OpenProcess שמקבל איזה הרשאות נרצה אליו ואת ה-PID שלו.

לאחר מכן נקצה מקום בזיכרון בגודל של ה-path ל-DLL שלנו (בדומה ל-malloc רק לתהליך חיצוני) בתהליך שנרצה להזריק אליו בעזרת הפונקצייה VirtualAllocEx, ונשמור במשתנה את הכתובת בזיכרון לשם. צריך לשים לב ש-WideCharacters תופסים שני בייטים בזיכרון ולכן נכפיל את הגודל ב-sizeof(WCHAR) ונוסיף 2 בייטים לבסוף שיהיו ה-null terminated bytes.

עכשיו נכתוב לתוך הזיכרון שהקצנו את ה-path ל-DLL בעזרת הפונקצייה WriteProcessMemory. ואז נשתמש בפונקצייה CreateRemoteThread כדי ליצור Thread חדש שירוץ על הפונקצייה LoadLibrary עם הפרמטר loc שהוא בעצם מצביע ל-path של ה-DLL שכתבנו קודם לזיכרון החיצוני.

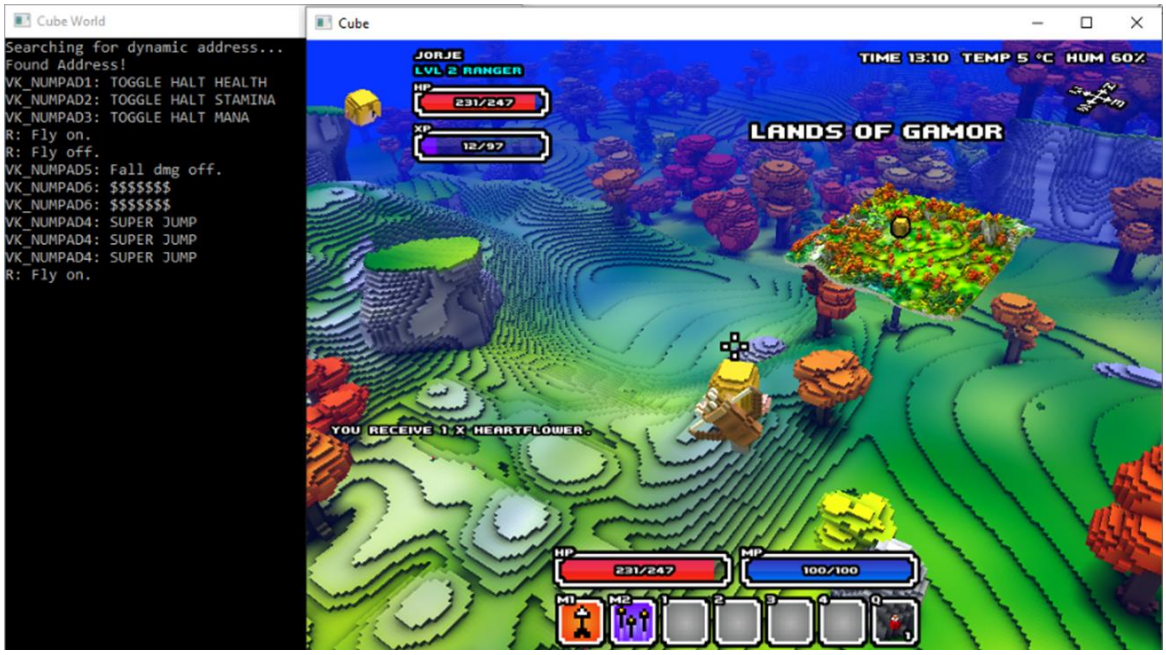
לאחר חלק זה בקוד ה-DLL יוזרק לתהליך כמו שרצינו:

```
/* Get handle to the process. */
HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, 0, procId);
if (hProc && hProc != INVALID_HANDLE_VALUE)
{
    /* Allocate memory inside the process. */
    LPVOID loc = VirtualAllocEx(hProc, 0, sizeof(WCHAR) * wcslen(dllPath) + 2, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    if (loc)
    {
        /* Write the dll's path inside the memory we allocated. */
        WriteProcessMemory(hProc, loc, dllPath, sizeof(WCHAR) * wcslen(dllPath) + 2, 0);

        /*
            Start a remote thread with the function LoadLibrary with the parameter loc
            Which is a pointer to the dll's path. It will load the dll now.
        */
        HANDLE hThread = CreateRemoteThread(hProc, 0, 0, (LPTHREAD_START_ROUTINE)LoadLibrary, loc, 0, 0);

        if (hThread)
        {
            CloseHandle(hThread);
        }
    }
    CloseHandle(hProc);
}
```

למי שמעוניין בהסבר מורחב על הטכניקה הזו מוזמן להיכנס לבלוג של אורי חדד:

https://oh-isecurity.blogspot.com/2020/02/dll-dll-injection-using-create-remote.html

אז כעת נבנה את הפרוייקט ל-DLL ונזריק אותו למשחק, יקפוץ לנו הקונסול ועכשיו נחכה שהמסלול שמצאנו יצביע לשחקן שלנו (כדאי לזוז טיפה במפה ותוך בערך חצי דקה הוא ימצא אותו, במאמר הבא אציג פיתרון אלגנטי יותר), ברגע שימצא יהיה כתוב לנו "Found Address!", ולאחר מכן נוכל להשתמש בצ'יט:



סיכום

מאמר זה הינו המאמר הראשון בסדרת המאמרים על הכנת צ'יטים למשחקים, במאמר זה הצגתי את התהליך הראשוני של בניית צ'יט פשוט למשחק מנקודת המבט שלי, בהתחלה מצאנו את המבנה של השחקן שלנו, לאחר מכן מצאנו את המצביע הסטטי לשחקן, ולבסוף הכנו לפי הפרטים שמצאנו את הצ'יט.

במהלך המאמר ראיתם פתרון לא אלגנטי למשל למצביע הסטטי לשחקן, במאמר הבא אציג איך בעזרת Reverse עם הכלי Ghidra ועם דיבאג על הפונקציות עם CheatEngine הגעתי גם למצביע סטטי לשחקן שלנו וגם רשימה מקושרת שמכילה בתוכה את כל הדמויות (מפלצות, חיות ואפילו שיחים) שנמצאות לידי. במאמר השלישי אציג איך עושים hook לפונקציות של DirectX ואיך משתמשים בהן כדי ליצור ESP (ריבועים על המסך שתוחמים כל דמות כרצוננו).

בחרתי במשחק זה כיוון שזה משחק ללא הגנות, לרוב המשחקים היום יש אנטי צ'יטים שיכולים להיות בסיסיים ויכולים להגיע לדרייברים שרצים ב-kernel למשל למי שמכיר המשחק LeagueOfLegends עובר לאנטי צ'יט מסוג זה:

https://na.leagueoflegends.com/en-us/news/dev/dev-null-anti-cheat-kernel-driver/

אני מקווה שבסוף סדרת המאמרים אפרסם מאמר על זה. תוכלו לפנות אליי לכל שאלה וייעוץ בנושא בכתובת המייל:

yoav.sh.j@gmail.com