

דייב מסוכן מאוד

מאת יונתן בר אור

הקדמה

מאמר זה מסכם פרויקט סוף שבוע שמטרתו היא לפתור סוגייה שהטרידה אותי במשך שנים רבות. תוצאות המחקר מסוכמות כאן בעברית, אך סוכמו גם באנגלית ב**בלוג הפרטי שלי** סוכמו בעתיד גם בקהילת ה-modding המתאימה.

סיפורנו מתחיל בשנות ה-90, תקופה זוהרת של דיסקטים, מודמים פרימיטיביים (אם בכלל), כפתורי Turbo, סאונד בלאסטר, והכי חשוב - משחקי DOS!

זוכרים את Commander Keen? מה לגבי Alley Cat או Sky Roads? ובכן, אני זוכר. למעשה, אני זוכר כל כך טוב שהאצבעות שלי זוכרות את התנועה המתאימה במשחקים רבים. הבעייה היא שמשחק בשם Dangerous Dave ("דייב המסוכן", נודע בעיקר פשוט כ-"דייב" או "דייב 1") לא יוצא לי מהראש, מכיוון שאחד מהבאגים הראשונים שאיי פעם גיליתי היו במשחק זה.

הייתי רוצה לציין לטובה אתר ישראלי בשם "[מסע על העבר](#)" המלווה אותנו כבר שנים רבות ומכיל משחקים מלאים להורדה, כולל משחקים ישראליים למהדרין ("בונוס", "גורדי" ואפילו כמה איזוטריים כגון "שוקי קווסט"). כמובן, גם דייב זמין להורדה בחינם. ניתן לשחק בהם בקלות באמצעות [DosBox](#), בו אשתמש מספר פעמים במאמר זה.

דייב המסוכן

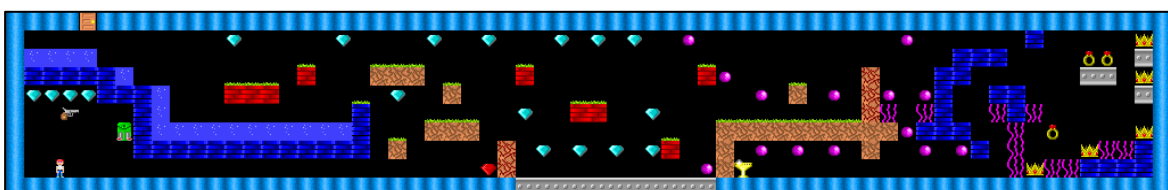
מה הסיפור עם דייב? מסתבר שיש עלילה כלשהי, אבל אני רוצה להתמקד במשחקיות - המשחק הוא משחק פלטפורמה, כלומר, יש דגש על קפיצה ממקום למקום. במשחק יש 10 שלבים, והמטרה בכל שלב היא פשוטה - לאסוף גביע ולעבור בדלת, כל זאת ללא להפגע ממים, אש, צמחים ארסיים או מפלצות.

הנה דוגמא נאה (שלב 4 הארור):



ישנם אלמנטים נוספים במשחק בנוסף לאלמנטים שכבר הזכרתי:

- א. איסוף חפצים. בכל שלב עלול להיות אקדח המאפשר לירות במפלצות. בנוסף, קיים Jetpack המאפשר ריחוף חופשי (אך עולה דלק).
 - ב. פסילות. בכל שלב ניתן להפסל, והמשחק פשוט יוריד את מספר הפסילות ("חיים") ב-1. כאשר מספר זה מגיע ל-0, זהו סוף המשחק.
 - ג. ניקוד: איסוף חפצים שונים (כגון יהלומים) מעלה את הניקוד. כל 20000 נקודות מעלות את מספר הפסילות ב-1 (אלא אם כן יש לנו כבר 3).
 - ד. שלבי בונוס. ישנם שלבים שבהם ניתן להגיע בצורה כלשהי לאזור "מעל התקרה" ואז לקפוץ מחוץ לשלב. כאשר זה קורה, מגיעים לשלב בונוס (ידוע במשחק בתור "Warp zone") שבדרך כלל מלא בחפצים יקרי ערך. סיום שלב בונוס מחזיר את דייב בחזרה לשלב שממנו הגענו.
- בתור ילד, ניסיתי לגלות את כל שלבי הבונוס. כבר מצאתי כאלה בשלבים 5, 8, 9 ו-10, ותהיתי האם ישנם עוד. ובכן, הנה צילום מסך של כל שלב 6:



הרעיון בשלב זה הוא פשוט יחסית - לנוע ימינה, לאסוף את הגביע ולהגיע לדלת באמצעות ה-Jetpack. הנקודה העדינה שגיליתי היא שכל עוד הגביע לא נאסף, המשחק מתייחס לדלת כאילו היא לא קיימת, ולכן אם לא אוספים את הגביע ניתן להגיע מעל לתקרה של שלב 6. קפיצה מעל התקרה שמאלה מביאה אותנו אל Warp Zone שנראה משוגע לחלוטין:



הקלטתי סרטון המתאר את הבאג, ניתן לצפות בו כאן:

<https://www.youtube.com/watch?v=95tPM7GGAel>

השאלה היא מה קורה כאן בדיוק היא השאלה שהטרידה אותי שנים רבות. בתקופה זו גדלתי, למדתי, האינטרנט הפך למה שהוא היום והעולם השתנה לחלוטין...

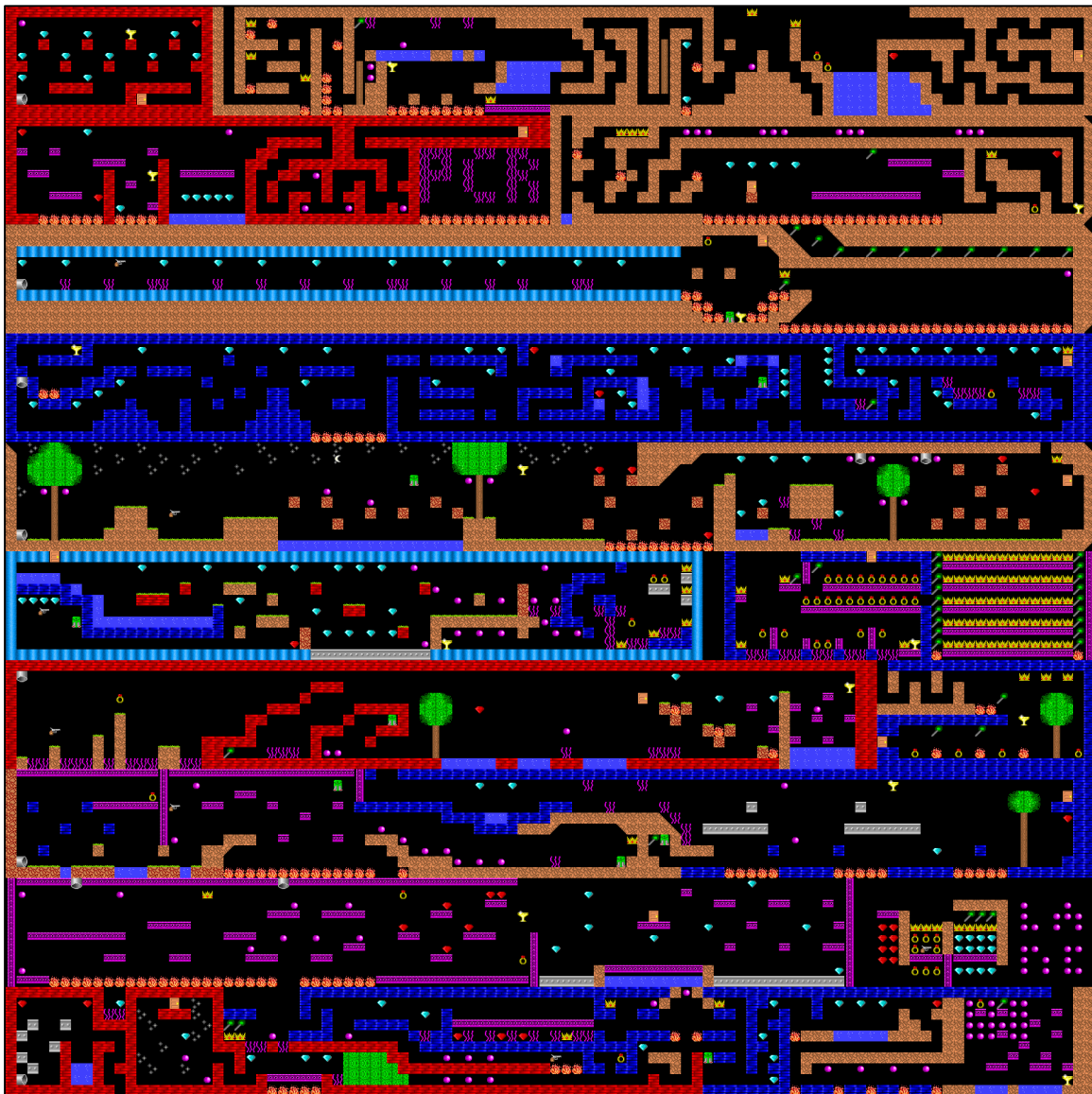
עד שבבוקר בהיר אחד החלטתי שהגיע הזמן לפתור את התעלומה!

עבודת בילוש ראשונית

האינסטינקט הראשוני של חוקרים לפעמים הוא לנסות לפתוח מיד את המשחק (שמגיע כקובץ יחיד, DAVE.EXE) ב-IDA או Ghidra ולהתחיל לעבוד. גישה זו איננה היעילה ביותר - הדבר הזול והאפקטיבי יותר הוא לקרוא, לחפש מידע שנאסף לפנינו ורק אז להתחיל אנליזה כבדה יותר. עם זאת, כבר כאשר הרצתי strings על DAVE.EXE גיליתי שאין כמעט מחרוזות סבירות. פתיחה של הבינארי ב-IDA (בלי באמת להתחיל לעבוד) חשפה כי הקובץ ככל הנראה packed. במקום להבין את אלגוריתם ה-packing בעצמי, החלטתי לחפש אונליין, ומהר הגעתי אל אתר modding מכובד בשם shikadi.net (אם אתם לא יודעים מה זה

Shikadi - אני ממליץ לשחק ב-Commander Keen). אתר זה הכיל מידע דיי מלא על משחקי עבר פופולריים, וביניהם גם על דייב, כולל מידע על אלגוריתם ה-packing (אלגוריתם LZW, למתעניינים), כלי לביצוע unpacking בשם UNLZEXE.EXE, ואפילו מידע על Off-sets שמושיים בקובץ, פורמט השלבים ועוד. ביצוע unpacking ל-DAVE.EXE מנפח את הקובץ ל-172848 בתים - בערך פי 2 מהקובץ המקורי.

השלב הבא היה להסתכל על מבנה השלבים - כיצד הם נשמרים בזכרון? קריאה באתר ה-modding חשפה כי שלבים שמורים בתור מערך גדול של בתים, כאשר כל בית הוא reference לתמונה להציג. אותה תמונה ידועה כ-tile, ושמורה אף היא בצורה packed (כן, מדובר על packing כפול). ה-tiles עצמם לא עניינו אותי כל כך, אבל מצאתי משתמש פעיל בשם MaiZure שעשה עבודה מצויינת הנוגעת ל-tiles והמפות ואפילו ניסה לשכתב את דייב ב-C\C++ שיריץ מעל SDL (עם הצלחה חלקית מאד). האתר של MaiZure מכיל כלים לחשיפת ה-tiles והמפות, ויכול אפילו להוציא את כל המפות לקובץ BMP! הנה התוצאה:





כפי שניתן לראות, ישנם 10 שלבים, אבל יותר מעניינת היא העובדה ששלבי ה-Warp zones "מתלבשים" על שלבים רגילים (למשל, שלב 2 מכיל את ה-Warp zone של שלב 5 בחלקו הימני). זוהי דרך מאד יעילה לשמור מידע תוך כדי שמירה על מבנה קבוע לכל שלב (100 על 10), ומייצגת באופן נאמן את החסכון בזיכרון של משחקים מאותה תקופה.

עכשיו כשהיו בידיי הכלים לביצוע unpacking וכלים לבחינת ה-tiles ומבנה השלבים, הגיע הזמן להמשיך במחקר!

מבנה השלבים

אתר ה-modding מתאר בצורה מצויינת כיצד בנוי כל שלב בזיכרון. ישנם 10 שלבים הנשמרים כמערך. כל שלב תופס בדיוק 1280 בתים בזיכרון, ומכיל:

א. מידע על אנימציה (של אויבים וכיו"ב) - 256 בתים. מבנה זה מעניין מאד, אך לא רלוונטי למחקר שלי ולכן התייחסתי אליו כקופסא שחורה.

ב. ה-tiles בשלב - 1000 בתים (כאמור, 100 על 10). מגיעים כ-references ל-tiles, מתחילים מהפינה השמאלית העליונה וממשיכים ימינה ולמטה.

ג. מידע לא משומש - 24 בתים.

ישנו שלב נוסף המכיל אך ורק tiles השמור בכתובת זיכרון אחרת - אותו שלב מייצג את מסך הפתיחה ותופס רק 70 בתים בזיכרון (הוא בגודל 10 על 7).

במקום אחר בזיכרון יש מערך בן 10 איברים הכולל את המידע הבא:

א. קואורדינטת ה-x בה דייב מתחיל את השלב.

ב. קואורדינטת ה-y בה דייב מתחיל את השלב.

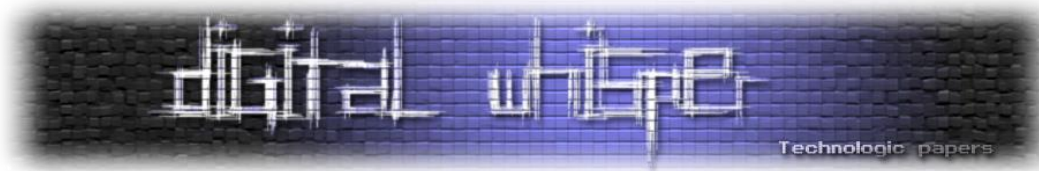
ג. התנועה ההתחלתית של דייב (נופל או נייח).

בנוסף, ישנו מידע הנוגע לשלבי הבונוס (המשולבים בשלבים הרגילים, כפי שראינו) הנשמר בשני מערכים בני 10 איברים כל אחד, ושני קבועים:

א. קואורדינטת ה-y בה דייב מתחיל כל שלב בונוס. זהו **קבוע** מכיוון שדייב מתחיל את כל שלבי הבונוס באותו גובה (הוא "נופל" במורד צינור, למתעניינים).

ב. התנועה ההתחלתית של דייב לכל שלב בונוס. זהו **קבוע** מכיוון שדייב מתחיל את כל שלבי הבונוס בנפילה.

ג. קואורדינטת ה-x בה דייב מתחיל את שלב הבונוס. מידע זה נשמר **במערך בן 10 איברים**, כלומר, לכל שלב יש קואורדינטה משלו.



מעבר לשלבי הבונוס

בשלב זה של המחקר החלטתי לחפש את המשתנה המכיל את השלב הנוכחי. עם DAVE.EXE ה-unpacked היו מלא מחרוזות שימושיות, ואחת מהן מציגה את הטקסט המופיע כאשר המשחק מסתיים בהצלחה, אחרי שלב 10:

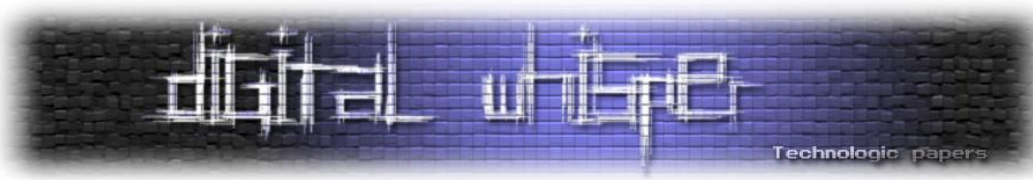
```
aCongratulation db '      congratulations!',0Dh,0Ah
db 0Ah
db 0Ah,0
aYouMadeItThrou db 'you made it through all the peril-',0Dh,0Ah,0
aOusAreasInClyd db 'ous areas in clyde',27h,'s hideout!',0Dh,0Ah
db 0Ah,0
aVeryGoodWorkDi db 'very good work! did you find',0Dh,0Ah,0
aThe4WarpZonesT db 'the 4 warp zones? they are located',0Dh,0Ah,0
aOnLevels589And db 'on levels 5,8,9 and 10. just jump',0Dh,0Ah,0
aOffTheTopOfThe db 'off the top of the screen at the',0Dh,0Ah,0
aExtremeLeftOrR db 'extreme left or right edge of the',0Dh,0Ah,0
aWorldAndVoilaY db 'world and voila! you',27h,'re there!',0Dh,0Ah
db 0Ah
db 0Ah,0
```

טקסט זה כמובן מחסל סופית את הרעיון ששלב הבונוס בשלב 6 היה מתוכנן, אבל יותר חשוב מזה - חיפוש cross-references לטקסט זה יראה לנו מתי טקסט זה מוצג. ההנחה שלי הייתה שישנה פונקציה שרצה בסוף כל שלב ובודקת כמה שלבים עברנו - אם עברנו 10 שלבים היא תציג את טקסט הניצחון, אחרת היא תעבור לשלב הבא. אכן מצאתי משהו שנראה ב-pseudo-code כך:

```
sub_1749B ();
word_56F4 = word_56F4 + 1;
sub_10BFB ();
if (word_573C == 1) {
    word_573C = 0;
    word_56F4 = word_6152;
}
if ((int)word_56F4 < 10) {
    sub_10BFB ();
    sub_14C69 ();
    sub_17631 ();
    sub_18CDA ();
    sub_14B9C ();
    sub_13235 ();
}
```

ובכן:

- ההשוואה ל-10 דיי ברורה: word_56F4 הוא המשתנה המייצג את מספר השלב.
- מעקב אחרי ה-flow של הקוד חשפה כי word_573C שומר האם אנחנו נמצאים בשלב בונוס או לא!
- שימו לב שאם בדיוק סיימנו שלב בונוס (word_573C == 1) אז השלב הנוכחי משוחזר (word_56F4 = word_6142), כלומר, word_6142 מתחזק את השלב ממנו הגענו! זה הגיוני מאד מכיוון ששלבי בונוס אינם שייכים לשלב המקור שלהם (למשל, שלב 5 הוא שלב "מלא" ושלב הבונוס שלו בכלל מופיע במפה של שלב 2).



כמובן, חלק מהמחקר כלל הרצה דינאמית ווידוא של כל ההנחות הללו. אף על פי שזו הייתה הפעם הראשונה בה הרצתי את הדיבאגר של DosBox, אני מרגיש צורך לציין שהוא היה אינטואיטיבי, אפקטיבי ועשיר מספיק לצורך דיבוג רציני:

```

DOS
FOR
DOSBox Debugger
---(Register Overview)---
EAX=0000002A  ESI=00000090  DS=26EF  ES=26EF  FS=0000  GS=0000  SS=26EF  Real
EBX=00000008  EDI=0000000A  CS=01A2  EIP=000034B4  C0 Z0 S0 00 A1 P0 D0 I1 T0
ECX=00000000  EBP=0000FFF2                                IOPL3  CPL0
EDX=000003DA  ESP=0000FFEA                                790280364
---(Data Overview  Scroll: page up/down)---
111C:22EE      00 2F 00 00 84 0F 00 00 02 20 00 00 86 02 00 00  ./.....
111C:22FE      02 20 00 2F 06 00 87 2F 00 20 00 02 00 00 02 04  . ./.../.
111C:230E      00 83 02 F0 00 20 06 00 89 F2 00 22 00 02 00 00  ....
111C:231E      02 00 02 02 00 85 02 F0 00 02 00 20 01 00 87 2F  .....
111C:232E      00 00 22 02 2F 00 02 00 00 81 20 02 02 00 85 02  .."/.....
111C:233E      20 00 02 00 28 01 00 8D FF F8 00 20 02 FF 00 20  ...(.
111C:234E      00 02 00 20 02 80 01 00 80 02 01 00 80 F8 00 00  ...
111C:235E      8E 02 22 F2 FF 02 F2 FF F2 20 00 0F 02 20 02 F0  ..".....
---(Code Overview  Scroll: up/down)---
01A2:34A8      D1E3                shl  bx,1
01A2:34AA      8BB7A601           mov  si,[bx+01A6]      ds:[01AE]=0090
01A2:34AE      8B1EF456           mov  bx,[56F4]        ds:[56F4]=0004
01A2:34B2      D1E3                shl  bx,1
01A2:34B4      8B876A01           mov  ax,[bx+016A]     ds:[0172]=0002
01A2:34B8      48                 dec  ax
01A2:34B9      A3F456            mov  [56F4],ax        ds:[56F4]=0004
01A2:34BC      E8AA17            call 00004C69 ($+17aa)
01A2:34BF      8B46FE            mov  ax,[bp-02]       ss:[FFF0]=002A
01A2:34C2      A3A057            mov  [57A0],ax        ds:[57A0]=0000
->
---(Variable Overview)---
---(OutPut/Input  Scroll: home/end)---
782934144: PIC:0 mask 0
782995896: PIC:0 mask 0
782996487: PIC:0 mask 0
783060504: PIC:0 mask 0
783061095: PIC:0 mask 0
783125040: PIC:0 mask 0
783125959: PIC:0 mask 0
783190494: PIC:0 mask 0
783191085: PIC:0 mask 0
783192362: PIC:0 mask 0
783192966: PIC:0 mask 0
784356818: PIT:PIT 0 Timer at 182.0818 Hz mode 3
787837637: PIT:PIT 0 Timer at 18.2065 Hz mode 3
DEBUG: Memory dump success.

```




לאחר וידוא ההנחות העבודה הפכה לקלה יותר. אם word_6142 מחזיק את השלב ממנו יצאנו, **כתיבה** אליו תתרחש בדיוק בשלב בו דייב עומד להכנס לשלב הבונוס. למזלנו, ישנה רק פונקציה אחת הכותבת על אותו משתנה, להלן pseudo-code רלוונטי:

```
g_curr_warp_zone_mapping = g_current_level;  
var2 = *(word *) (g_current_level * 2 + 0x192);  
var3 = *(word *) (g_current_level * 2 + 0x1a6);  
g_current_level = *(int *) (g_current_level * 2 + 0x16a) - 1;  
sub_14C69();  
g_start_y = 0x10;
```

כפי שניתן לראות, קפיצה לתוך שלב בונוס מגבה את השלב הנוכחי ב-word_6142 (כבר שיניתי את שמו ל-g_curr_warp_zone_mapping). לאחר מכן, השלב החדש שאליו קופצים נשמר בתוך מערך המתחיל ב-0x16A המאונדקס על ידי השלב הנוכחי, כל אלמנט בו תופס שני בתים (מכיוון שמדובר בארכיטקטורת 16 ביט, 2 בתים הוא גודל "סטנדרטי"). מכאן ברור שאותו מערך מתחזק את המיפוי בין קפיצה משלב רגיל לשלב בונוס!

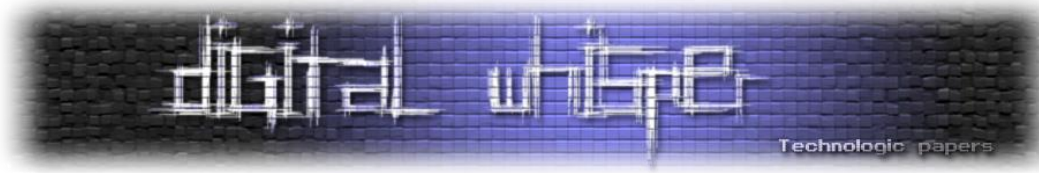
כאשר מתבוננים בערכים השמורים במערך זה ומתוך הנחה שהמערך מכיל 10 איברים, ניתן לראות את הערכים: 0, 0, 0, 0, 2, 0, 0, 6, 7, 1. כפי שניתן לראות, האיבר החמישי הוא 2, ואכן שלב הבונוס של שלב 5 ממומש בשלב 2! כמובן, האפסים עבור השלבים האחרים ככל הנראה מציינים שלא אמור להיות שלבי בונוס עבורם.

בנוסף נשים לב שהשלבים הם 0-based אך הערכים במערך הם 1-based (לכן האיבר החמישי במערך הוא 2 ולא 1), ולכן יש חיסור של 1 בחישוב מספר המפה שאליה קופצים.

שלב הבונוס של שלב 6

הבה נדמיין יציאה מהמסך של שלב 6 - כפי שראינו, האיבר השישי במערך המיפוי מכיל 0, אך מכיוון שיש חיסור של 1, השלב הנוכחי יהיה מינוס 1, או במילים יפות - 0xFFFF (שימו לב ששוב לעובדה שזו ארכיטקטורת 16-ביט).

תודות לקהילת ה-modding, אנחנו יודעים את הכתובת בה שלבים מתוארים בזיכרון (כאמור, כ-1280 בתים כל אחד). ובכן, הוספת 0xFFFF לאותה כתובת גורמת לשלב להטען מכתובת אחרת לחלוטין! הבתים של אותה כתובת זיכרון מפורשים כבתים של שלב, כולל הצגת ה-tiles המתאימים. בנוסף לכך שהמשחק מפרש בתים שלא נועדו לכך כשלב לגיטימי, חלק מה-tiles אינם תקינים (מספרי ה-tiles אמנם מתחילים מאפס אבל יש פחות מ-256 כאלה), דבר המסביר את הפיקסלים המוזרים שנראים בחלק התחתון של המסך



כאשר הבאג מתרחש. הוספתי את "שלב הבונוס" ל-parser שכתבתי, וייצגתי tiles לא ממופים כסימני שאלה:

```
Buggy warp level:
##XXX/ \XXX\W =====
MX&JXW *
X XXXX # # $ #
* W JXXX
| ##X _ |* XXX - - XXX*
| ##X # X X XX/ XX * *
| ##X> _ _ _ _ _XX XXXXX* *
| ###XXXXXoXXXoXXXoXXXoXXXXXXXXXXXXXXXXXXXXXXXXX WXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX###WWW#####WWW#####WWW#####?M_ ? ?T? ??? Xg]??M! W WXW?WGW?WGW?WGW??=?
??G??G?? ? ? 3?????? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ????
????????????? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? L?0 ? ?_!
WWW|||---g ??????????????LT|-----V-----V-----
$ | |* * M Q $$ $ -
| | |* * * * $ --
```

תהיות נוספות

לאחר פרסום המחקר הראשוני, קיבלתי לא מעט הודעות בנושא, כולל חוויות נוסטלגיה שאנשים רצו לחלוק איתי, אבל גם שאלות נוספות. למשל, כאשר עולים על התקרה של שלב 6 אבל נעים ימינה ולא שמאלה, לא מגיעים לאותו שלב בונוס מקולל, אלא מגיעים לשלב הבונוס של שלב 8! ובכן, זה דווקא ברור - הבונוס של שלב 8 נשמר בחלק הימני של שלב 6, ולכן קפיצה ימינה אפילו לא יוצאת מגבולות השלב - המשחק מפרש זאת כאילו אנחנו עדיין בשלב 6. למעשה, ניתן להרוויח מכך - לאסוף את כל היהלומים מאותו שלב בונוס של שלב 8 בשלב 6 ואז להגיע לשלב הבונוס של שלב 8, כלומר, ניתן לאסוף אותם פעמיים במשחק.

תהיה נוספת שעלתה על ידי **cp77fk4r** דורשת התייחסות רצינית יותר:

מסתבר שכאשר מגיעים ל-99,999 נקודות, כל חזרה מהתפריט הראשי של המשחק מעלה את כמות החיים ב-1 (אלא אם כן הם כבר היו מלאים). זה נשמע כמו באג נוסף ששווה לבחון, ומכיוון שהידע היה טרי עדיין במוחי, החלטתי להשקיע ערב נוסף במחקר שיענה על שאלה זו.

ובכן, מכיוון שאני כבר יודע מהו המשתנה ששומר את מספר השלב, מצאתי פונקציית אתחול שנקראת כאשר המשחק מתחיל, ושם ראיתי השמה של הערך הקבוע 3 למשתנה. מכיוון שאני יודע שהמשחק מתחיל עם 3 חיים, קל היה להבין שמשנתנה זה מתחזק את כמות החיים של דיב.

לאחר דיבוג נוסף ושינוי שמות משתנים, פונקציית האתחול נראית בערך ככה:

```
g_lives = 3;
g_score_lo = 0;
g_score_hi = 0;
g_next_goal_lo = 0;
g_next_goal_hi = 0;
g_is_game_over = 0;
g_current_level = 0;
g_some_level_reference = 0;
g_maybe_levels_left = 10;
g_is_warp_zone = 0;
```

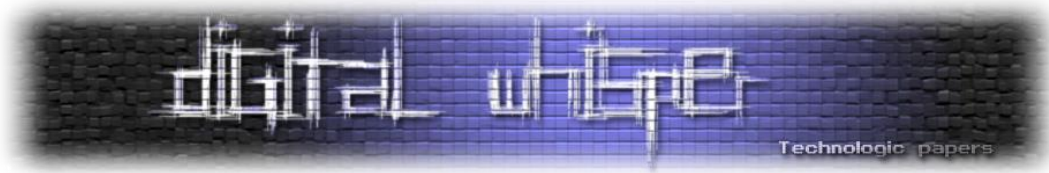
נשים לב שישנם שני משתנים ששומרים את הניקוד הנוכחי - זה הגיוני כי הניקוד המקסימאלי (99,999 נקודות) גדול מ-0xFFFF, ולכן הניקוד יוצג ב-4 בתים. באופן דומה, ישנם זוג משתנים השומרים את הניקוד הבא שכאשר מגיעים אליו מקבלים חיים נוספים. ביצוע cross-references לאותם משתנים עזר לי להגיע לפונקציה שבודקת מתי להעלות את החיים לדייב, והנה pseudo-code רלוונטי:

```
if ((g_score_hi - g_next_goal_hi != (uint)(g_score_lo < g_next_goal_lo)) ||
    (0x4e20 < g_score_lo - g_next_goal_lo)) {
    // ...
    g_next_goal_lo = g_score_lo;
    if ((int)g_lives < 3) {
        UpdateSprite(g_lives * 0x10 + 0x100, 0,
                    *(int *) (g_some_base * g_some_offset * 2 + 0x216));

        g_lives = g_lives + 1;
        PlaySound(0xc);
    }
}
// ...
if ((g_score_hi != 0) && ((1 < g_score_hi || (0x869f < g_score_lo)))) {
    g_score_lo = 0x869f;
    g_score_hi = 1;
}
```

ישנם פרטים רבים כאן:

- נתחיל דווקא מהסוף - אפשר לראות שהערך המקסימאלי של הניקוד הוא 99999. מדוע? ובכן, $99999 = 0x1869f$, ולכן החלק הגבוה יהיה תמיד 1 והחלק הנמוך יהיה $0x869f$. לכן, לא ניתן אף פעם לעבור מגבלה זו.
- אם נדלג על השורה הראשונה, השורה השנייה היא די ברורה - היא בודקת האם המרחק בין החלק התחתון של הניקוד שלנו ובין החלק התחתון של הניקוד הבא לחיים גדול מ- $0x4e20$, או בדצימאלית - 20,000. זוהי הסיבה שכל 20,000 מקבלים חיים חדשים.



• ניתן לראות כי מספר החיים המקסימאלי הוא 3 - רק אם מספר החיים הנוכחי קטן ממש מ-3 מעלים אותו ב-1.

השורה הראשונה היא הסיבה להתנהגות המוזרה הזו. ההנחה שלי היא ששורה זו נועדה לטפל במצב בו הניקוד הנוכחי עבר את הסף לחיים, אבל הדרך שבה זה נעשה מתייחס לשני החלקים בנפרד (החלק העליון והתחתון). ובכן, דיבוג חשף כי לאחר ניקוד של 99,999, ה-next goal הוא 100,000, ולכן במצב זה:

- החלק העליון של הניקוד (g_score_hi) הוא 1.
 - החלק התחתון של הניקוד (g_score_lo) הוא 0x869f.
 - החלק העליון של הסף (g_next_goal_hi) הוא 1.
 - החלק התחתון של הסף (g_next_goal_lo) הוא 0x86a0.
- לכן, התנאי בשורה הראשונה תמיד מתקיים (כי החלק השמאלי הוא 0 והחלק הימני תמיד 1 בגלל המגבלה של הניקוד המקסימאלי).

מדוע מקבלים חיים נוספים כאשר חוזרים מהתפריט הראשי? ובכן, אותה פונקציה הבודקת האם צריך להעלות את החיים נקראת כאשר יש פגיעה בחפצים (הגיוני, כי אז הניקוד בדרך כלל עולה) אבל נקראת גם כאשר חוזרים מהתפריט הראשי. אני מניח שהסיבה לכך היא כדי לצייר את הניקוד מחדש, אבל בכל מקרה - כאשר מגיעים לניקוד המקסימאלי, החיים יכולים לעלות גם בחזרה מהתפריט אבל גם כאשר פוגעים בחפצים.

סיכום

העליתי את תוצאות המחקר שלי לבלוג הפרטי שלי:

https://github.com/yo-yo-yo-jbo/dangerous_dave

כולל את ה-parser שלי שמאפשר עכשיו גם לערוך שלבים וכיו"ב:



למרות שמדובר בפרוייקט חובבני לסוף השבוע, היה לי כיף מאד לעבוד עליו! סיכום המחקרים מאתר ה-modding (shikadi.net) חסך לי זמן רב, ואני מודה מאד לקהילה ולחוקריה.

אף על פי שכיום כמעט ואין טעם לעשות Reverse-Engineering ל-Intel real mode (אולי מלבד להסתכל על MBR שלצערי עדיין רלוונטי איכשהו בשנת 2023) אבל אני עדיין חושב שזה מחקר מעניין. ספציפית, הדרך שבה משחקי DOS ישנים ניסו לחסוך כל טיפה של זיכרון בצורה מתוחכמת היא אמנות שלצערי גוועת לאט (נסו למצוא משחקים מודרניים ששוקלים פחות מ-2 מגה, למשל).

מוזמנים לעקוב אחריי בטוויטר:

https://twitter.com/yo_yo_yo_jbo

ובבלוג שלי:

<https://yo-yo-yo-jbo.github.io>

בו אני מפרסם פה ושם.

תודה על הקריאה!