

The Fire Ball Game

פרויקט בשפת אסמבלי

אלון דיין י'5

מוגש ליהודה אור, הכפר הירוק

יוני 2019



תוכן עניינים

1	שער
2	תוכן עניינים
3	מבוא
4	רפלקציה
5	המשחק שלי
7	חלק תאורטי
10	חלק מעשי
17	קוד המשחק

מבוא

בעבודה זו אציג לכם, הקוראים הנכבדים, את הפרויקט שלי בשפת אסמבלי שכתבתי במהלך שנת הלימודים הנוכחית.

בספר זה אתאר לכם את תחושותיי במהלך הפרויקט ואת הקשיים שניצבו בדרכי. אסביר לכם בקצרה על שפת האסמבלי ועל פקודות בסיסיות בשפה זו.

בנוסף, אציג בפניכם את הפרויקט שלי, אסביר על המבנה שלו ואסביר לפרטים חלק אחד ממנו. לבסוף, אצרף את הקוד המלא לפרויקט שלי.

רפלקציה

כתיבת הפרויקט הייתה תהליך משמעותי מאוד בשבילי. השקעתי זמן רב בכתיבתו והשקעתי בו מאמץ רב.

בעקבות כתיבת הפרויקט רכשתי כמה וכמה מיומנויות חשובות: בעקבות הפרויקט למדתי איך ללמוד בכוחות עצמי. במהלך הפרויקט נתקלתי במספר בעיות שאותן לא ידעתי לפתור. על מנת להתגבר על בעיות אלו נאלצתי לחפש תשובות באינטרנט ותוך כדי כך, ללמוד דברים חדשים על שפת האסמבלי. בסופו של דבר, הצלחתי לפתור את הבעיות שנתקלתי בהן בכוחות עצמי ושיפרתי את היכולות האוטודידקטיות שלי.

בזכות פרויקט זה שיפרתי את יכולות התכנות שלי. במהלך שנה זו למדתי לראשונה לכתוב קוד בשפת אסמבלי. מלבד למידת השפה, במהלך השנה רכשתי יכולות חשובות לכתיבת קוד. למדתי לחשוב על אלגוריתמים וכיצד לכתוב פתרונות שנחשבים פשוטים בשפת עילית אך קשים בהרבה לכתיבה בשפת סף כמו אסמבלי. יתר על כן, שפת אסמבלי לימדה אותי על מבנה המחשב והמעבד, דבר מאוד חשוב לעתיד שלי כמתכנת. בנוסף, למדתי על כלי הדיבאגר שעזר לי לפתור הרבה מאוד בעיות בפרויקט ואפילו בשפות אחרות.

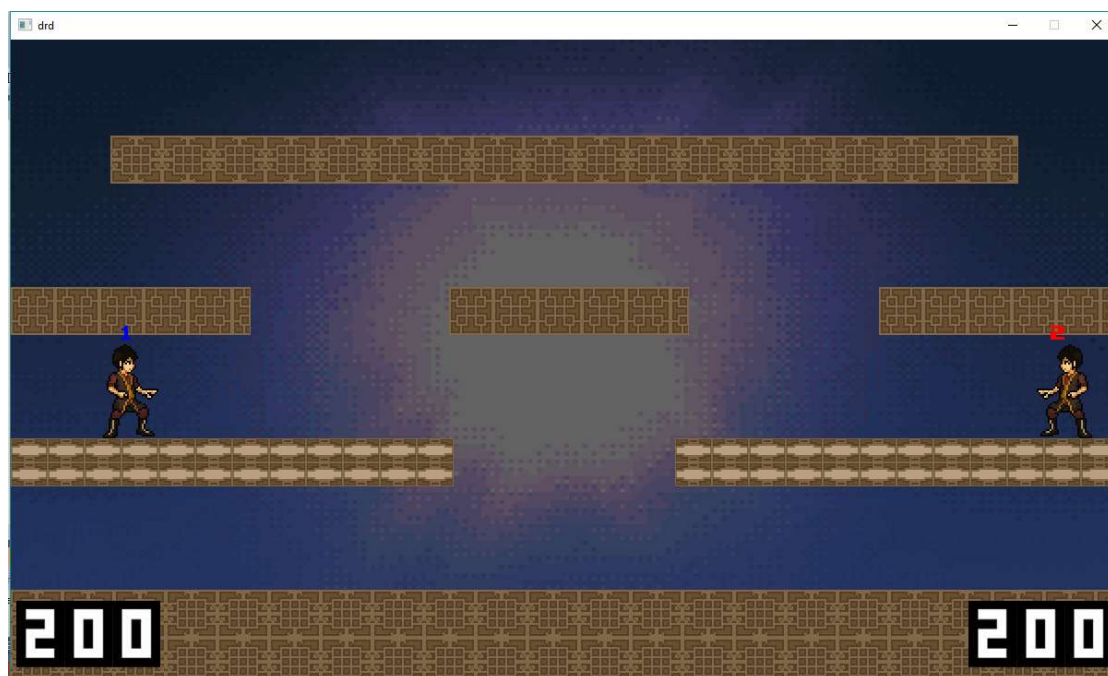
בזמן הפרויקט, שיפרתי את הדרך בה אני חושב על כתיבת קוד. עד עתה, כאשר הייתי צריך לכתוב קוד בשפה עילית הייתי חושב על התוכנית הכללית שלי ומתחיל לכתוב אותו מההתחלה לסוף. לעומת זאת, כאשר רציתי לכתוב קוד באסמבלי, גיליתי ששיטה זו אינה יעילה ובאמצע כתיבת הקוד גיליתי טעויות ב"תכנית" ההתחלתית שלי. לכן, די מהר בתחילת כתיבת הפרויקט, אימצתי שיטה חדשה שעזרה לי בהמשך עבודתי בכתיבת קוד.

בכל שלב של כתיבת הקוד, חשבתי מראש על ה"שלד" של הקוד שלי בחלק זה, חילקתי את הקוד למספר פונקציות כך שיהיה יותר מסודר ובכל פעם כתבתי פונקציה אחת, עם הרבה הערות בתוכה. שיטה זו גרמה לתחושה של הצלחה בגלל שתהליך כתיבת הקוד היה מורכב מהרבה הצלחות קטנות לפני השלמת שלב בכתיבה.

נהנתי מאוד מכתיבת הפרויקט ולמידת השפה. בזמן התהליך ניצבו מולי מספר קשיים אך הצלחתי להתגבר על כולם, והגעתי לתוצאה שרציתי מלכתחילה.

המשחק שלי

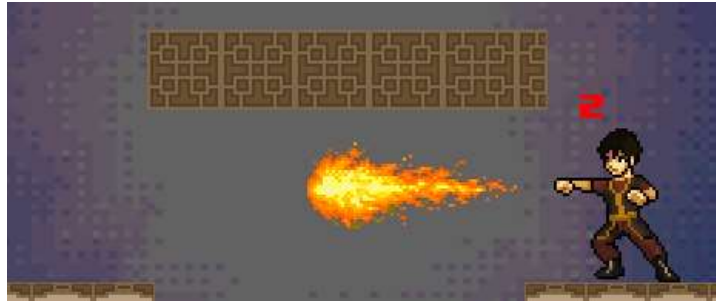
המצב ההתחלתי של המשחק – שני השחקנים בתוך הזירה ומוצגות נקודות הפגיעה ההתחלתיות של שניהם.



שחקן מספר 1 בעת קפיצה.



שחקן מספר 2 בעת ירייה.



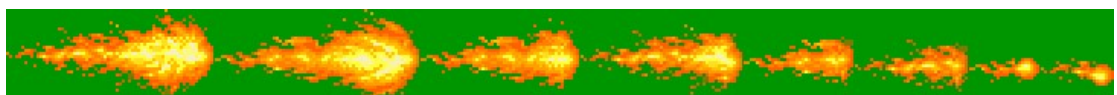
שחקן מספר 2 בעת ריצה



Player 1 won!
Press R to play again

מסך הניצחון לשחקן 1.

רצף התמונות לאנימצית הירייה של כדור האש



חלק תאורטי:

הפרויקט כתוב בשפה אסמבלי 32 ביט (MASM 32), שהיא שפת סף שפותחה ע"י חברת "אינטל". שפה זו היא שפת סף, כלומר, הקוד עצמו מתורגם ישירות לבינארית (רצף של הספרות 0 ו-1 בכמויות שונות) והמחשב קורא את הבינארית ומבצע את הקוד. זאת בניגוד לשפות אחרות שעוברות עוד מספר תהליכים עד שהן מתורגמות לבינארית.

מושגים בסיסיים:

הזיכרון מורכב משני חלקים:

- **הזכרון הקשיח (Hard Disk/Hard Drive)** – מקום האחסון הראשי של המחשב. זכרון זה הוא מכאני ולכן המידע נשאר גם כאשר המחשב כבוי. יש בו מקום זכרון רב.

- **זכרון גישה אקראית (RAM)** – מקום האחסון הזמני של המחשב. זכרון זה הוא אלקטרוני ומהיר יותר מהזכרון הקשיח. בכדי להשתמש בתמונה יש להעביר אותה לזכרון זה.

הזכרון מורכב ממספר עצום של בתים המסודרים יחדיו. ניתן לגשת לכל בית (רק ב-RAM) באמצעות הכתובת שלו, מעין מספר סידורי, שמייצגת את המיקום של הבית.

בית (byte) – יחידת זכרון המורכבת מ-8 סיביות. בית אחד יכול לייצג מספר מאפס עד 255, או תו בודד.

סיבית (bit) – ספרה בינארית שיכולה לייצג אפס או 1. זוהי יחידת הזכרון הקטנה ביותר במחשב. בבינארית אפשר לייצג את המספר $(2^n - 1)$ בבסיס 10 באמצעות n סיביות.

Nibble – חצי בית, מורכב מ-4 סיביות. יכול לייצג מספר מאפס עד 15. נהוג להשתמש בכמות זו של זכרון בשביל לייצג ספרה אחת בספירה בבסיס הקסדצימאלי (hex), שפה בה יש 15 ספרות (0-9 ואז a-f).

word – אוסף של שני בתים, או 16 ביטים.

dword – אוסף של שני מילים, או 4 בתים.

אוגר (register) – חלק מהמעבד שמשמש כתא לאחסון נתונים. ניתן לעשות עליו פעולות חשבוניות שונות ולהעביר אליו ערכים מהזכרון. לאוגרים יכולים להיות גדלים שונים (לרוב בין 16 ל-64 סיביות).

דוגמאות לאוגרים: eax, ebx, ecx, edx, edi, esi (כל אלו הם אוגרים בגודל 32 ביט).

פקודות באסמבלי

mov – פקודה המעבירה גודל מסוים (מאוגר/זכרון/פרמטר של פונקציה) לאוגר/זכרון אחר. לדוגמה: `mov eax, ebx` = העבר את הערך שיש בתוך ebx לתוך eax.

add – פקודה המגדילה אוגר/גודל בזכרון בגודל אחר. לדוגמה: `add eax, 3` = הגדל את הערך שיש בתוך eax ב-3.

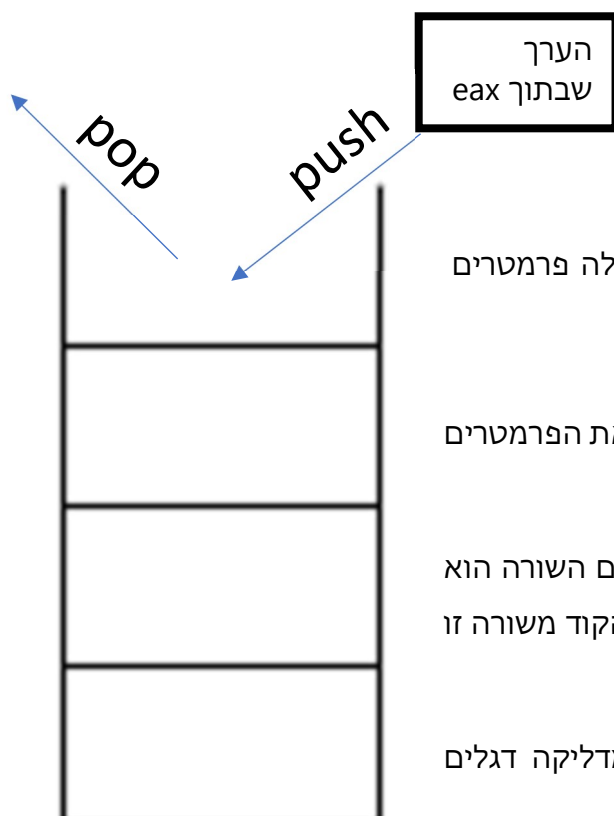
sub – פקודה המקטינה אוגר/גודל בזכרון בגודל אחר. לדוגמה: `sub var, eax` = הקטן את המשתנה var בערך של eax.

inc – פעולה המגדילה גודל בזכרון/אוגר באחד. לדוגמה: `inc ecx` = הגדל את ecx באחד.

dec – פעולה המקטינה גודל בזכרון/אוגר באחד. לדוגמה: `sub edx` = הקטן את edx באחד.

div – פעולה המחלקת את הצמדת האוגרים eax:edx בפרמטר המתקבל. התוצאה נשמרת ב-eax והשאריית נשמרת ב-edx. לדוגמה אם `eax=4, ebx=7, edx=0` אז הפקודה `div ebx` תגרום לכך: `edx = 3, eax = 1`.

pop, push – הפקודות האלו קשורות לפעולת **המחסנית**. המחסנית היא טיפוס נתונים מסוג LIFO (Last In First Out), כלומר, האיבר האחרון שנכנס למחסנית יצא ראשון. המחסנית משמשת כאמצעי להעברת משתנים לפונקציות ושמירה על משתנים זמניים.



הפקודה `push eax` תכניס למחסנית את הערך שבתוך eax.

הפקודה `pop ebx` תכניס את הערך האחרון במחסנית לתוך ebx.

Invoke – פקודה הקוראת לפונקציה ומעבירה לה פרמטרים מהזכרון או באמצעות אוגרים.

למשל: `invoke test1, eax, 3`
הפקודה הזו תקרא לפונקציה test1 ותעביר לה את הפרמטרים eax ו-3.

LINE: - הכרזה על שם של שורה (במקרה זה, שם השורה הוא LINE). ניתן לקפוץ לשורה זו, כלומר להריץ את הקוד משורה זו והלאה.

cmp – הפקודה הזו משווה בין שני גדלים ומדליקה דגלים המתארים את יחס הגדלים בניהם.

jmp LINE – קפוץ לשורה LINE
jl LINE – קפוץ לשורה LINE אם קטן.
je LINE – קפוץ לשורה LINE אם שווה.
jg LINE – קפוץ לשורה LINE אם גדול.
jne LINE – קפוץ לשורה LINE אם לא שווה.
jge LINE – קפוץ לשורה LINE אם גדול או שווה.
jle LINE – קפוץ לשורה LINE אם קטן או שווה.

ret – בדרך כלל משתמשים בפקודה זו בסוף של פונקציה. פקודה זו מחזירה את הקוד לרוץ מהנקודה הקודמת בה קרא לפונקציה.

.code – שורה זו מצהירה כי משורה והלאה, זהו החלק שבו נכתוב את ההוראות לתוכנה.

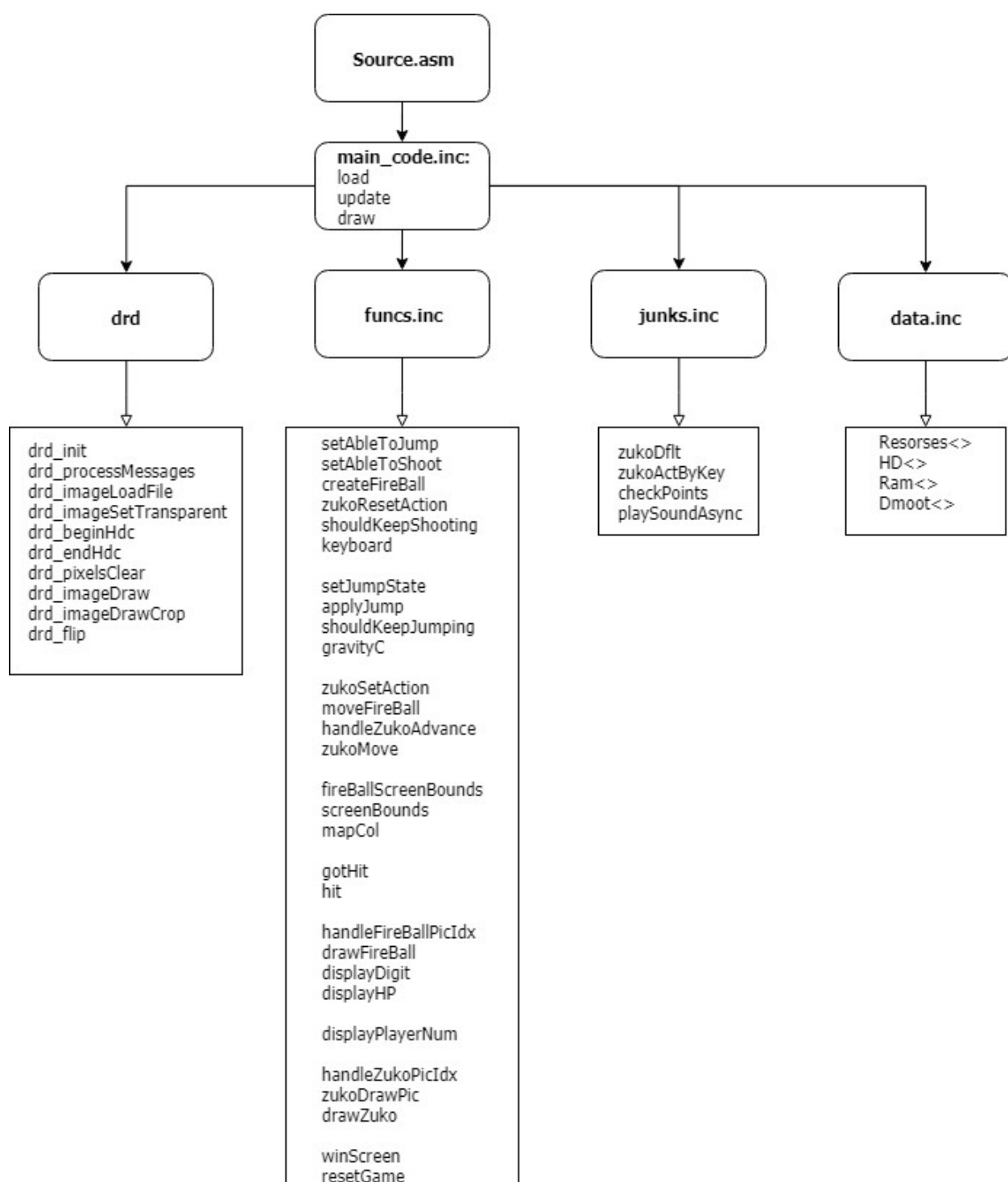
.data – שורה זו מצהירה כי משורה זו והלאה, זהו החלק שבו נצהיר על משתנים. משתנים יכולים להיות מסוג byte, word ו-dword או מבני נתונים מורכבים יותר כגון מערך או struct אחר שהמתכנת יוצר.

[] – הסוגריים המרובעות משמשות בשביל לגשת לערך כלשהו בכתובת בזכרון. לדוגמה, אם ב-eax יש את הכתובת של בית כלשהו, בשביל לגשת לערך שלו עלינו לכתוב [eax].

offset – כאשר כותבים offset לפני שם של משתנה, מקבלים את הכתובת של משתנה זה בזכרון. למשל HP offset מייצג את הכתובת של HP בזכרון.

חלק מעשי:

תרשים הקבצים והפונקציות



הסבר על הפונקציות

drd – קובץ שמכיל פעולות לציור גרפי על המסך

drd_init – פותחת חלון.

drd_processMessages – מאפשרת לחלון להסגר בעת הרצת המשחק.

drd_imageLoadFile – טוענת תמונה מתוך קובץ וממלאת את ה- שלה.

drd_imageSetTransparent – קובעת את הצבע שיקבע כשקוף לתמונה.

drd_beginHdc – הסבר בהמשך.

drd_endHdc – הסבר בהמשך.

drd_pixelsClear – מוחקת את כל התמונות מהמסך.

drd_imageDraw – מציירת תמונה על ה-buffer.

drd_imageDrawCrop – מציירת חלק מתמונה על ה-buffer.

drd_flip – מציירת על המסך את כל התמונות שצויירו על ה-buffer.

funcs.inc – קובץ המכיל את רוב הפונקציות של המשחק

setAbleToJump – בודקת האם הדמות יכולה לקפוץ (בהתחשב האם היא כבר קופצת או עומדת על קרקע).

setAbleToShoot – בודקת האם הדמות יכולה לירות (בהתחשב האם היא קופצת, נופלת או כבר יורה).

createFireBall – משמיעה קטע שמע של כדור אש ויוצרת כדור בכיוון הירייה של הדמות.

zukoResetAction – מאתחלת פעולה של דמות על פי הפרמטר של הפונקציה (מאתחלת את המשתנים שקשורים לאנימציות הפעולה).

shouldKeepShooting – בודקת האם הדמות צריכה להפסיק לירות, ליצור כדור אש או להפסיק לירות.

keyboard – קולטת לחיצת מקשים בהתאם לצורך.

setJumpState – קובעת שהדמות קופצת.

applyJump – גורמת לדמות לקפוץ ובודקת מתי יש להפסיק את הקפיצה.

shouldKeepJumping – בודקת האם הדמות צריכה להמשיך לקפוץ.

gravityC – גורמת לדמות ליפול כאשר היא לא קופצת או עומדת על קרקע.

zukoSetAction – קובעת האם הדמות שינתה את כיוונה וקובעת מה הפעולה שהיא מבצעת כרגע (עמידה, ריצה, קפיצה, נפילה וירייה).

moveFireBall – מקדמת את כדור האש אם הוא מופיע.

handleZukoAdvance – בודקת האם צריך לקדם את הדמות או לא (בשביל להאט את המשחק בעת הצורך).

zukoMove – מזיזה את הדמות ואת כדור האש שלה.

fireBallScreenBounds – בודקת האם כדור האש יצא מגבולות המסך ומחזירה אותו למסך.

screenBounds – בודקת האם הדמות יצאה מגבולות המסך ומחזירה אותה למסך אם כן.

mapCol – בודקת האם הדמות זזה לתוך אובייקט במסך מצדדיה.

gotHit – משמיעה קטע שמע של פגיעה, מורידה לדמות נקודות פגיעה ובודקת האם הדמות מתה.

hit – בודקת האם הדמות נפגעה מכדור אש.

handleFireBallPicIdx – מטפלת באינדקס האנימציה של כדור האש. במשך כל קריאה היא בודקת האם עליה להעלות אותו כך שהתמונה הבאה תוצג.

drawFireBall – מציירת את כדור האש.

displayDigit – מציגה ספרה אחת במיקום מסוים (הפרמטרים הם הספר והמיקום).

displayHP – מציגה את נקודות הפגיעה של הדמות.

displayPlayerNum – מציגה את מספר השחקן מעל הדמות.

handleZukoPicIdx – מטפלת באינדקס האנימציה של הדמות בהתאם לפעולה אותה מבצעת הדמות באותו רגע. לכל פעולה יש מונה שקטן בכל קריאה של הפונקציה. אם המונה מתחלק בקבוע (שונה לכל פעולה) אז אינדקס האנימציה של הדמות מתקדם והתמונה הבאה ברצף האנימציה מוצגת על המסך. כאשר המונה של הפעולה הוא אפס, מאפסים את המונה של הפעולה ואת אינדקס האנימציה.

zukoDrawPic – פונקציית עזר שמציירת תמונה של דמות במיקום מסוים על פי משתנים רבים שהיא מקבלת.

drawZuko – מציירת את הדמות, כדור האש שלה, נקודות הפגיעה שלה ומספר השחקן שלה. היא מציירת את הדמות על פי הפעולה שהיא עושה באותו רגע והכיוון אליו הדמות פונה.

winScreen – מציירת את מסך הנצחון על פי הדמות שניצחה.

resetGame – מאפסת את המשחק בכדי שיתחיל מחדש.

junks.inc – קובץ המכיל מספר פונקציות עזר כלליות

zukoDflt – מאתחלת את המהירות של הדמות לאפס (בכל תחילת מחזור של הפונקציות draw ו-update).

zukoActByKey – בודקת האם המקש שהתקבל נלחץ. אם כן, מכניסה את ערך כלשהו למיקום מסוים בזכרון. (המיקום בזכרון, הערך והמקש הם פרמטרים של הפונקציה).

playSoundAsync – משמיעה קטע שמע שהתקבל.

checkPoints – בודקת האם אחד הצבעים שהתקבלו מופיע במערך הנקודות שהתקבלו.

data.inc – קובץ המכיל את המשתנים בזכרון

Img<> - מבנה נתונים המוגדר בתוך קובץ drd. מייצג תמונה כאשר היא בתוך ה-RAM.

Resources<> - מבנה נתונים המכיל כמה תמונות (Img<>) ואת הנתונים שלהם (path).

HD<> - מכיל את הנתונים של תמונות הדמות וכדור האש.

Ram<> - מכיל את התמונות (Img<>) של הדמות וכדור האש.

Dmoot<> - מכיל את המשתנים הקשורים לדמות (מיקום, כיוון, מקשים, נקודות פגיעה, פעולה נוכחית ועוד).

הסבר על פונקציה – checkpoints

```
1 checkpoints proc adrObj:dword, adrArr:dword, sizeofArr:dword, color1:dword,
  color2: dword
2 ;                                     offset of arr[0], max bytes in arr , bgr, bgr
3
4 ;result in eax -> 0-colors don't show, 1- one or both colors show
```

הפונקציה מקבלת את המיקום של הדמות בזכרון, את המיקום של מערך הנקודות בזכרון, את גודלו של המערך (בבתים) ושני צבעים (הצבעים מיוצגים באופן הבא - 0x00bbggr).

הנקודה של
מיקום
הדמות

הפונקציה מחזירה את תוצאתה באוגר eax. אם אף אחד מהצבעים לא הופיע בכל הנקודות, היא תחזיר אפס. אם לפחות אחד הצבעים הופיע לפחות פעם אחת, הפונקציה תחזיר 1.

שימוש אחד לפונקציה זו הוא בפונקציית gravityC. במקום לשמור את מיקום הקרקע בכל המסך, קוראים לפונקציה checkpoints ובודקים האם הפיקסלים שמתחת לרגלי הדמות הם בצבע של הקרקע.

נקודות הבדיקה הן חיבור של ערכי הנקודות במערך המתקבל ונקודת המיקום של הדמות.

```
invoke drd_beginHdc;get handle
mov esi, eax ; esi = handle
mov ecx,0;ecx = idx
```



נקודת
הבדיקה
הראשונה

נקודת
הבדיקה
השנייה

בכדי לבדוק את צבע הפיקסלים בתוך חלון ה-drd נצטרך לקבל את ה-handle של החלון, מעין קוד סידורי שמאפשר לתוכניות אחרות לגשת אל נתוני החלון. בשביל לקבל את ה-handle זה עלינו לזמן את הפונקציה drd_beginHdc שמעבירה את ה-handle ל-eax. בנוסף, בתום השימוש ב-handle עלינו לזמן את הפונקציה drd_endHdc ולתת לה את ה-handle של החלון.

```

dloop:
    mov ebx, adrObj

    mov edi, (Dmoot PTR [ebx]).pos.x
    mov edx, (Dmoot PTR [ebx]).pos.y

    push ecx
    push esi

    mov esi, ecx
    add esi, adrArr; esi=adrArr+ idx
    add edi, [esi]; point[idx].x
    add edx, [esi + 4]; point[idx].y
    ;; edi = ctrlPt1X, edx = ctrlPt1Y
    pop esi; hdc = esi
    mov eax, esi ; hdc = eax
    invoke GetPixel, eax, edi, edx ; eax = pixel
    cmp eax, color1
    je SHOW
    cmp eax, color2
    je SHOW

    pop ecx; ecx = idx
    add ecx, 8
    cmp ecx, sizeofArr
    jlt dloop; if(ecx+8<sizeofArr) stay in loop

```

כעת, נתחיל את הלולאה.

נשים בתוך האוגרים edi,edx את ערכי ה-x,y של הדמות (בהתאמה).

נשמור במחסנית את ecx (האינדקס הנוכחי במערך) ואת esi (ה-handle של חלון ה-drd).

נוסיף לערכי ה-x,y של הדמות את ערכי הנקודה הנוכחית במערך.

נחזיר ל-esi את ה-handle ונשמור אותו גם ב-eax.

נקרא ל-GetPixel בערכי ה-x,y הנוכחים ונשווה את הצבע שיש בו (מוחזר בתוך eax) לשני הצבעים שקיבלנו כפרמטרים.

במקרה שהצבעים לא מופיעים,

נחזיר את האינדקס של המערך ל-ecx ונוסיף לו 8 (><POINT מורכב משני dword ולכן גודלו הוא 8 בתים). נבדוק האם הגענו לאיבר האחרון במערך ואם לא, נקפוץ לתחילת הלולאה ונחזור על התהליך עד שנסיים לעבור על כל הנקודות במערך.

```

NOTSHOW:
    mov eax, 0
    invoke drd_endHdc, esi
    ret

SHOW:
    invoke drd_endHdc, esi
    mov eax, 1
    ret

```

אם אחד הצבעים מופיע, נקפוץ לשורה SHOW ובה נחזיר את הערך 1.

אם הלולאה הסתיימה ושני הצבעים לא הופיעו, הקוד ממשיך לשורה NOTSHOW ובה הפונקציה מחזירה 0.

בשני המקרים, נקרא לפונקציה drd_endHdc לפני סיום הפונקציה.

הפונקציה בשלמותה:

```
1  checkPoints proc adrObj:dword, adrArr:dword, sizeofArr:dword, color1:dword,
    color2: dword
2  ;                                     offset of arr[0], max bytes in arr , bgr, bgr
3
4  ;result in eax -> 0-colors don't show, 1- one or both colors show
5
6      invoke drd_beginHdc;get handle
7      mov esi, eax ; esi = handle
8      mov ecx,0;ecx = idx
9      dloop:
10         mov ebx, adrObj
11
12         mov edi, (Dmoot PTR [ebx]).pos.x
13         ;mov edi, [ebx+DM.pos.x]
14         mov edx, [ebx+DM.pos.y]
15
16         push ecx
17         push esi
18
19         mov esi, ecx
20         add esi, adrArr;esi=adrArr+ idx
21         add edi, [esi];point[idx].x
22         add edx, [esi + 4];point[idx].y
23         ;; edi = ctrlPt1X, edx = ctrlPt1Y
24         pop esi;hdc = esi
25         mov eax, esi ;hdc = eax
26         invoke GetPixel, eax, edi,edx ; eax = pixel
27         cmp eax, color1
28         je SHOW
29         cmp eax, color2
30         je SHOW
31
32         pop ecx; ecx = idx
33         add ecx,8
34         cmp ecx, sizeofArr
35         jl dloop;if(ecx+8<sizeofArr) stay in Loop
36
37     NOTSHOW:
38     mov eax,0
39     invoke drd_endHdc, esi
40 ret
41
42     SHOW:
43     invoke drd_endHdc, esi
44     mov eax,1
45 ret
46 checkPoints endp
```



```
1  include \masm32\include\masm32rt.inc
2  include main_code.inc
3  .code
4
5  main proc
6      invoke load
7      loopi:
8          invoke update
9          invoke draw
10     jmp loopi
11 ret
12 main endp
13
14 end main
```

```
1 include drd.inc
2 includelib drd.lib
3 include data.inc
4 include junks.inc
5 include funcs.inc
6 .code
7
8 load proc
9     invoke drd_init,1200,700,0;;open window
10    ;;load images
11    invoke drd_imageLoadFile,offset res.bgpath1,offset res.bg1
12    invoke drd_imageLoadFile,offset res.digitsPath,offset res.digits
13
14    invoke drd_imageLoadFile,offset res.win1Path,offset res.win1
15    invoke drd_imageLoadFile,offset res.win2Path,offset res.win2
16
17    invoke drd_imageLoadFile,offset res.dig1Path,offset res.dig1
18    invoke drd_imageSetTransparent,offset res.dig1,0ffffffh
19
20    invoke drd_imageLoadFile,offset res.dig2Path,offset res.dig2
21    invoke drd_imageSetTransparent,offset res.dig2,0ffffffh
22
23    invoke drd_imageLoadFile, offset hd.zuko.standR, offset ram.zuko.standR
24    invoke drd_imageSetTransparent,offset ram.zuko.standR,0009600h
25
26    invoke drd_imageLoadFile, offset hd.zuko.standL, offset ram.zuko.standL
27    invoke drd_imageSetTransparent,offset ram.zuko.standL,0009600h
28
29    invoke drd_imageLoadFile, offset hd.zuko.runR, offset ram.zuko.runR
30    invoke drd_imageSetTransparent,offset ram.zuko.runR,0009600h
31
32    invoke drd_imageLoadFile, offset hd.zuko.runL, offset ram.zuko.runL
33    invoke drd_imageSetTransparent,offset ram.zuko.runL,0009600h
34
35    invoke drd_imageLoadFile, offset hd.zuko.fallR, offset ram.zuko.fallR
36    invoke drd_imageSetTransparent,offset ram.zuko.fallR,0009600h
37
38    invoke drd_imageLoadFile, offset hd.zuko.fallL, offset ram.zuko.fallL
39    invoke drd_imageSetTransparent,offset ram.zuko.fallL,0009600h
40
41    invoke drd_imageLoadFile, offset hd.zuko.jumpR, offset ram.zuko.jumpR
42    invoke drd_imageSetTransparent,offset ram.zuko.jumpR,0009600h
43
44    invoke drd_imageLoadFile, offset hd.zuko.jumpL, offset ram.zuko.jumpL
45    invoke drd_imageSetTransparent,offset ram.zuko.jumpL,0009600h
46
47    invoke drd_imageLoadFile, offset hd.zuko.shootR, offset ram.zuko.shootR
48    invoke drd_imageSetTransparent,offset ram.zuko.shootR,0009600h
49
50    invoke drd_imageLoadFile, offset hd.zuko.shootL, offset ram.zuko.shootL
51    invoke drd_imageSetTransparent,offset ram.zuko.shootL,0009600h
52
53    invoke drd_imageLoadFile, offset hd.fireBall.moveR, offset
54        ram.fireBall.moveR
55    invoke drd_imageSetTransparent,offset ram.fireBall.moveR,0009600h
```

```
56     invoke drd_imageLoadFile, offset hd.fireBall.moveL, offset      ↗
       ram.fireBall.moveL
57     invoke drd_imageSetTransparent,offset ram.fireBall.moveL,0009600h
58 ret
59 load endp
60
61 update proc
62     invoke drd_processMessages;;allow window to be closed
63
64     ;;check if someone won the game
65     cmp win, 0
66     jne SOMEONEWON
67
68     invoke zukoDflt, offset zuko1
69     invoke zukoDflt, offset zuko2
70
71     invoke keyboard, offset zuko1
72     invoke keyboard, offset zuko2
73
74     invoke gravityC, offset zuko1
75     invoke gravityC, offset zuko2
76
77     invoke zukoSetAction, offset zuko1
78     invoke zukoSetAction, offset zuko2
79
80     invoke mapCol, offset zuko1
81     invoke mapCol, offset zuko2
82
83     invoke zukoMove, offset zuko1
84     invoke zukoMove, offset zuko2
85
86     invoke screenBounds, offset zuko1
87     invoke screenBounds, offset zuko2
88
89     invoke hit, offset zuko1, offset zuko2
90     invoke hit, offset zuko2, offset zuko1
91 ret
92
93     SOMEONEWON:
94     ;;skip update
95     invoke zukoActByKey,0,offset Rkey,offset win, 0
96     cmp win,0
97     je RESET
98 ret
99
100    RESET:
101    invoke resetGame, offset zuko1
102    invoke resetGame, offset zuko2
103 ret
104 update endp
105
106 draw proc
107     invoke drd_pixelsClear,0;;clear screen
108
109     ;;check if someone won
110     cmp win,0
```

```
111     jne SOMEONEWON
112
113     invoke drd_imageDraw,offset res.bg1,0,0
114
115     invoke drawZuko, offset zuko1
116     invoke drawZuko, offset zuko2
117
118     invoke drd_flip
119 ret
120
121     SOMEONEWON:
122     invoke winScreen, offset zuko1, offset zuko2
123     invoke drd_flip
124 ret
125 draw endp
```

```
1  .const
2  ;;structs
3  HDzuko struct
4      standR byte "zuko/standR.png",0
5      standL byte "zuko/standL.png",0
6      runR byte "zuko/runR.png",0
7      runL byte "zuko/runL.png",0
8      jump byte "zuko/jump.png",0
9      fallR byte "zuko/fallR.png",0
10     fallL byte "zuko/fallL.png",0
11     jumpR byte "zuko/jumpR.png",0
12     jumpL byte "zuko/jumpL.png",0
13     shootR byte "zuko/shootR.png",0
14     shootL byte "zuko/shootL.png",0
15 HDzuko ends
16
17 HDfireBall struct
18     moveR byte "otherPics/fireBallR.png",0
19     moveL byte "otherPics/fireBallL.png",0
20     fireSound byte "sound/fireBall.wav",0
21     hitSound byte "sound/hit.wav",0
22 HDfireBall ends
23
24 HD struct
25     zuko HDzuko<>
26     fireBall HDfireBall<>
27 HD ends
28
29 AnimfireBall struct
30     moveR Img<>
31     moveL Img<>
32 AnimfireBall ends
33
34 Anim struct
35     standR Img<>
36     standL Img<>
37     runR Img<>
38     runL Img<>
39     fallR Img<>
40     fallL Img<>
41     jumpR Img<>
42     jumpL Img<>
43     shootR Img<>
44     shootL Img<>
45 Anim ends
46
47 Ram struct
48     zuko Anim<>
49     fireBall AnimfireBall<>
50 Ram ends
51
52 Resorses struct
53     bgpath1 byte "otherPics/newFullBg.png",0
54     bg1 Img<>
55     digitsPath byte "otherPics/digits.png",0
56     digits Img<>
```

```

57     win1Path byte "otherPics/player1won.png",0
58     win1 Img<>
59     win2Path byte "otherPics/player2won.png",0
60     win2 Img<>
61     dig1Path byte "otherPics/dig1.png",0
62     dig1 Img<>
63     dig2Path byte "otherPics/dig2.png",0
64     dig2 Img<>
65 Resorses ends
66
67 Keys struct
68     left dword ?
69     right dword ?
70     up dword ?
71     shoot dword ?
72 Keys ends
73
74 Action struct
75     ogCount dword ?
76     count dword ?
77     cycle dword ?
78     ;ogCount/cycle = number of frames in sprite sheet
79     ;ogCount = cycle*number_of_frames_in_sprite_sheet
80 Action ends
81
82 fireBall struct
83     pos POINT<0,0>
84     drcX dword 0
85     isShowing dword 0; 0- not showing, 1- showing
86     fireAnim Action<400,400,50>
87     picIdx dword 0;;idx in Anim
88     lastW dword 0
89 fireBall ends
90
91 Dmoot struct
92     pos POINT<>
93     startPos POINT<>;the position Dmoot starts at the begining of the game
94     drc POINT<>
95     keys Keys<>
96     adrUpPts dword ?
97     maxBytsUpPts dword ?
98     adrLeftPts dword ?
99     maxBytsLeftPts dword ?
100    adrRightPts dword ?
101    maxBytsRightPts dword ?
102
103    startPtHP POINT<>; the place to draw the HP at
104    lossCode dword ?
105    playerNum dword ?
106
107    toPlayerNum POINT <15,-22>
108
109    moveAct Action<4,4,2>
110    standAnim Action<100,100,50>
111    runAnim Action<200,200,25>
112    shootAnim Action<200,200,50>

```

```

113     fallAnim Action<100,100,50>
114     jumpAnim Action<80,80,25>
115
116
117     picIdx dword 0;;idx in animation
118     facing dword 0; 0-right, 1-left
119     actionOn dword 0 ;; 0-stand, 1-run, 2-jump, 3-fall, 4-shooting
120     jumpState dword 0 ;; 0-pre, 1-jump
121     advance dword 1 ;; 0-don't, 1-do
122
123     ogJumpH dword 204
124     jumpH dword 204 ;; at that height starts to fall after jump
125     ogJumpFreeze dword 50
126     jumpFreeze dword 50;;freeze before jump for that amount of time
127
128     ctrlPt1 POINT<0,102>;fall
129     ctrlPt2 POINT<76,102>;fall
130
131     ball fireBall<>
132     HP dword 200
133 Dmoot ends
134
135 .data
136
137 emptyDword dword 0
138 Rkey dword VK_R
139
140 res Resorses<>
141 hd HD<>
142 ram Ram<>
143
144 win dword 0 ;; 0-game, 2-zuko2 won, 1-zuko1 won
145
146 StartHP dword 200
147
148 UpPts POINT 2 dup(<0,-1>,<66,-1>)
149 LeftPts POINT 3 dup(<-1,0>,<-1,52>,<-1,101>)
150 RightPts POINT 6 dup(<76,0>,<76,52>,<76,101>,<66,0>,<66,52>,<66,101>)
151
152 zuko1 Dmoot<<100,300>,<100,300>,<0,0>,<VK_LEFT,VK_RIGHT,VK_UP,VK_DOWN>,offset
    UpPts,SIZEOF UpPts, offset LeftPts, SIZEOF LeftPts, offset RightPts, SIZEOF
    RightPts,<110,608>,2,1>;sizeof - number of bytes
153 zuko2 Dmoot<<1110,300>,<1110,300>,<0,0>,<VK_A,VK_D,VK_W,VK_S>,offset
    UpPts,SIZEOF UpPts, offset LeftPts, SIZEOF LeftPts, offset RightPts, SIZEOF
    RightPts,<1142,608>,1,2>;sizeof - n
154
155 groundColor dword 000486880h
156 errorColor dword 000ffffffh
157 blackColor dword 000000000h
158
159 zukoH dword 102
160 zukoW dword 76
161
162 zukoStandWid dword 62
163 zukoStandRsrcX dword 2 dup(0,78)
164 zukoStandLsrcX dword 2 dup(78,0)

```

```
165
166 zukoRunRW dword 8 dup(77,88,82,64,77,86,82,66)
167 zukoRunRsrcX dword 8 dup(0,78,167,250,315,393,480,563)
168 zukoRunLW dword 8 dup(66,82,86,77,64,82,88,77)
169 zukoRunLsrcX dword 8 dup(1,68,151,238,316,381,464,553)
170
171 zukoShootRW dword 4 dup(64,54,82,82)
172 zukoShootRsrcX dword 4 dup(0,65,120,203)
173 zukoShootLW dword 4 dup(64,54,82,82)
174 zukoShootLsrcX dword 4 dup(222,167,84,1)
175
176 zukoFallRW dword 4 dup(76,76,66,66)
177 zukoFallRsrcX dword 4 dup(0,77,154,221)
178 zukoFallLW dword 4 dup(66,66,76,76)
179 zukoFallLsrcX dword 4 dup(1,67,134,211)
180
181 zukoPreJumpRW dword 58
182 zukoPreJumpRsrcX dword 1
183 zukoJumpRW dword 8 dup(78,78,68,68,68,68,78,78)
184 zukoJumpRsrcX dword 8 dup(59,138,217,286,286,217,138,59)
185 zukoPostJumpRW dword 3 dup(58,65,58)
186 zukoPostJumpRsrcX dword 3 dup(355,414,480)
187
188 zukoPreJumpLW dword 58
189 zukoPreJumpLsrcX dword 481
190 zukoJumpLW dword 8 dup(78,78,68,68,68,68,78,78)
191 zukoJumpLsrcX dword 8 dup(402,323,254,185,185,254,323,402)
192 zukoPostJumpLW dword 3 dup(58,65,58)
193 zukoPostJumpLsrcX dword 3 dup(126,60,59)
194
195 ;flags for jump State
196 PRE_JUMP dword 0
197 WHILE_JUMP dword 1
198
199 START_FIREBALL dword 50
200
201 ctrlPts POINT 2 dup(<0,102>,<56,102>)
202
203 createBallR POINT <83,11>
204 createBallL POINT <-136,11>
205
206 ballH dword 60
207 fireBallRW dword 8 dup(136,134,104,106,72,74,46,48)
208 fireBallRsrcX dword 8 dup(0,137,272,377,484,557,632,679)
209 fireBallLW dword 8 dup(136,134,104,106,72,74,46,48)
210 fireBallLsrcX dword 8 dup(591,456,351,244,171,96,49,0)
211 ballDamage dword 50
212
213 digitH dword 72
214 digitW dword 52
215 digitSrcX dword 10 dup(0,52,104,156,208,260,312,364,416,468)
```



```
1  .code
2
3  setAbleToJump proc adrObj: dword
4  ;;result in eax: 1-can jump, 0- cannot jump
5  ;; cannot jump if falling or jumping
6      mov ebx, adrObj
7      mov ecx, (Dmoot PTR [ebx]).actionOn
8      cmp ecx, 3 ;fall
9      je CANNOT
10     cmp ecx, 2 ;;jump
11     je CANNOT
12
13     ;can jump
14     mov eax,1
15 ret
16
17     CANNOT:
18     mov eax,0
19 ret
20
21 setAbleToJump endp
22
23 setAbleToShoot proc adrObj:dword
24 ;;result in eax: 1-can jump, 0- cannot jump
25     mov ebx, adrObj
26     mov ecx, (Dmoot PTR [ebx]).actionOn
27     cmp ecx, 3 ;fall
28     je CANNOT
29     cmp ecx, 2 ;;jump
30     je CANNOT
31     cmp ecx, 4;shooting
32     je CANNOTSH
33
34     CAN:
35     mov eax, 1
36 ret
37
38     CANNOT:
39     mov eax,0
40 ret
41
42 CANNOTSH:
43     mov eax,0
44 ret
45 setAbleToShoot endp
46
47 createFireBall proc adrObj:dword
48     invoke playSoundAsync, offset hd.fireBall.fireSound
49
50     ;zuko1.ball = fireBall
51     mov ebx, adrObj
52     ASSUME ebx: PTR Dmoot
53     mov edx, [ebx].ball.fireAnim.ogCount
54     mov [ebx].ball.fireAnim.count, edx
55     mov [ebx].ball.picIdx, 0
56
```

```
57     mov eax, [ebx].pos.x
58     mov ecx, [ebx].pos.y
59
60     mov edx, [ebx].facing
61     cmp edx, 1
62     je LeftPt
63     cmp edx, 0
64     je RightPt
65
66     retPt:
67     mov [ebx].ball.pos.x, eax
68     mov [ebx].ball.pos.y, ecx
69
70     mov edx, [ebx].facing
71     cmp edx, 1
72     je Left
73     cmp edx, 0
74     je Right
75
76     RET_DRC:
77     mov [ebx].ball.drcX, edx
78
79     mov edx, 1
80     mov [ebx].ball.isShowing, edx
81     ASSUME ebx: nothing
82     ret
83
84     RightPt:
85     add eax, createBallR.x
86     add ecx, createBallR.y
87     jmp retPt
88
89     LeftPt:
90     add eax, createBallL.x
91     add ecx, createBallL.y
92     jmp retPt
93
94     Right:
95     mov edx, 2
96     jmp RET_DRC
97
98     Left:
99     mov edx, -2
100    jmp RET_DRC
101    createFireBall endp
102
103    zukoResetAction proc adrObj: dword, act: dword
104        mov eax, adrObj
105        ASSUME eax: PTR Dmoot
106        mov [eax].picIdx, 0;reset PicIdx
107        mov ebx, act
108
109        cmp ebx, 0
110        je ResetStand
111        cmp ebx, 1
112        je ResetRunAnim
```

```
113     cmp ebx, 3
114     je ResetFallAnim
115     cmp ebx, 2
116     je ResetJumpAnim
117     cmp ebx, 4
118     je ResetShootAnim
119
120     RETURN:
121     ASSUME eax: nothing
122 ret
123
124     ResetStand:
125     ASSUME eax: PTR Dmoot
126     mov edx, [eax].standAnim.ogCount
127     mov [eax].standAnim.count, edx
128 jmp RETURN
129
130     ResetRunAnim:
131     ASSUME eax: PTR Dmoot
132     mov edx, [eax].runAnim.ogCount
133     mov [eax].runAnim.count, edx
134 jmp RETURN
135
136     ResetFallAnim:
137     ASSUME eax: PTR Dmoot
138     mov edx, [eax].fallAnim.ogCount
139     mov [eax].fallAnim.count, edx
140 jmp RETURN
141
142     ResetJumpAnim:
143     ASSUME eax: PTR Dmoot
144     mov edx, [eax].jumpAnim.ogCount
145     mov [eax].jumpAnim.count, edx
146 jmp RETURN
147
148     ResetShootAnim:
149     ASSUME eax: PTR Dmoot
150     mov edx, [eax].shootAnim.ogCount
151     mov [eax].shootAnim.count, edx
152 jmp RETURN
153 zukoResetAction endp
154
155 shouldKeepShooting proc adrObj:dword
156     mov ebx, adrObj
157     mov edx, START_FIREBALL
158     cmp (Dmoot PTR [ebx]).shootAnim.count, edx
159     je CREATEFIREBALL
160     cmp (Dmoot PTR [ebx]).shootAnim.count, 0
161     je STOP_SHOOTING
162 ret
163
164     STOP_SHOOTING:
165     invoke zukoResetAction, adrObj, 4
166     mov ebx, adrObj
167     mov (Dmoot PTR [ebx]).actionOn, 0
168 ret
```

```
169
170     CREATEFIREBALL:
171     invoke createFireBall, adrObj
172 ret
173 shouldKeepShooting endp
174
175 keyboard proc adrObj: dword
176     mov ebx, adrObj
177
178     mov edx, 4
179     cmp (Dmoot PTR [ebx]).actionOn, edx
180     je WHILE_SHOOT
181
182     invoke zukoActByKey, adrObj, Dmoot.keys.left, Dmoot.drc.x, -1
183     invoke zukoActByKey, adrObj, Dmoot.keys.right, Dmoot.drc.x, 1
184
185     invoke setAbleToJump, adrObj
186     cmp eax, 0
187     je CANNOTJUMP
188
189     ;;can jump
190     invoke zukoActByKey, adrObj, Dmoot.keys.up, Dmoot.drc.y, -1
191
192     CANNOTJUMP:
193     ;;empty jump
194     mov edx, ebx
195     add edx, Dmoot.keys.up
196     invoke zukoActByKey, 0, edx, offset emptyDword, 0
197
198     invoke setAbleToShoot, adrObj
199     cmp eax, 0
200     je CANNOTSHOOT
201
202     ;;can shoot
203     invoke zukoActByKey, adrObj, Dmoot.keys.shoot, Dmoot.actionOn, 4
204     mov edx, 4
205     cmp (Dmoot PTR [ebx]).actionOn, edx
206     je FIRST_SHOOT
207
208     CANNOTSHOOT:
209     ;;empty shoot
210     mov edx, ebx
211     add edx, Dmoot.keys.shoot
212     invoke zukoActByKey, 0, edx, offset emptyDword, 0
213 ret
214
215     FIRST_SHOOT:
216 ret
217
218     WHILE_SHOOT:
219     invoke shouldKeepShooting, ebx
220 ret
221 keyboard endp
222
223 setJumpState proc adrObj: dword, jmpStt: dword
224     mov ebx, adrObj
```

```

225     mov ecx, jmpStt
226     mov (Dmoot PTR [ebx]).jumpState, ecx
227 ret
228 setJumpState endp
229
230 applyJump proc adrObj: dword
231     mov ebx, adrObj
232     mov edx, PRE_JUMP
233     cmp (Dmoot PTR [ebx]).jumpState, edx
234     je FREEZE
235     NO_MORE_FREEZE:
236     ;;change drc
237     mov (Dmoot PTR [ebx]).drc.y, -1
238 ret
239
240     FREEZE:
241     cmp (Dmoot PTR [ebx]).jumpFreeze, 0
242     je PastFreeze
243     ;;jumpFreeze--
244     mov edx, 1
245     sub (Dmoot PTR [ebx]).jumpFreeze, edx
246 ret
247
248     PastFreeze:
249     ;;reset jump freeze
250     mov edx, (Dmoot PTR [ebx]).ogJumpFreeze
251     mov (Dmoot PTR [ebx]).jumpFreeze, edx
252     ;;change jumpState
253     mov edx, WHILE_JUMP
254     mov (Dmoot PTR [ebx]).jumpState, edx
255     jmp NO_MORE_FREEZE
256 applyJump endp
257
258 shouldKeepJumping proc adrObj:dword
259 ;; result in actionOn
260     mov ecx, adrObj
261
262     mov ebx, (Dmoot PTR [ecx]).adrUpPts
263     mov esi, (Dmoot PTR [ecx]).maxBytsUpPts
264     invoke checkPoints, adrObj, ebx, esi, groundColor, errorColor
265
266     mov ecx, adrObj
267     ASSUME ecx: PTR Dmoot
268     cmp eax, 1; colors showing
269     je StopJump
270
271     ;;check jumpH
272     cmp [ecx].jumpH, 0
273     jle StopJump
274
275     ;;check out of frame
276     cmp [ecx].pos.y, 0
277     jle StopJump
278
279     ;;if ok then actionOn = 2, jumpState = 1,
280     mov edx, WHILE_JUMP

```

```

281     mov [ecx].jumpState, edx
282
283     ASSUME ecx: NOTHING
284 ret
285
286     StopJump:
287     mov ecx,adrObj
288     ASSUME ecx: PTR Dmoot
289     mov [ecx].actionOn, 3 ;;change actionOn
290     mov ebx, [ecx].ogJumpH;;reset jumpH
291     mov [ecx].jumpH, ebx
292     ASSUME ecx: NOTHING
293 ret
294 shouldKeepJumping endp
295
296 gravityC proc adrObj: dword
297     mov eax, adrObj
298     mov ebx, 2
299     cmp (Dmoot PTR [eax]).actionOn, ebx
300     je JUMPING
301
302     mov ebx, -1
303     cmp (Dmoot PTR [eax]).drc.y, ebx
304     je STARTJUMPING
305
306     ASSUME eax: nothing
307     invoke checkPoints , adrObj, offset ctrlPts, 16, groundColor, errorColor
308     cmp eax, 0
309     je FALLING
310 ret
311
312     JUMPING:
313     mov edx, PRE_JUMP
314     cmp (Dmoot PTR [eax]).jumpState, edx
315     jne WHILEJUMPING
316     ;;not moving at all
317     invoke applyJump, adrObj
318 ret
319
320     WHILEJUMPING:
321     ;;skip gravity
322     invoke shouldKeepJumping, adrObj
323     mov eax, adrObj
324     cmp (Dmoot PTR [eax]).actionOn, 2 ;;if not jumping then apply gravity
325     jne FALLING
326     ;;jumping
327     invoke applyJump, adrObj
328 ret
329
330     STARTJUMPING:
331     mov ebx, 2
332     mov (Dmoot PTR [eax]).actionOn, ebx
333     ;;apply jump
334     invoke setJumpState, adrObj, PRE_JUMP
335     invoke applyJump, adrObj
336     ;;skip gravity
337 ret

```

```
337     FALLING:
338     mov eax, adrObj
339     mov edx, 1
340     mov (Dmoot PTR [eax]).drc.y, edx
341     ret
342 gravityC endp
343
344 zukoSetAction proc adrObj: dword
345     mov edi, adrObj
346     mov ecx, (Dmoot PTR [edi]).actionOn ; ecx = lastAction
347     mov esi, (Dmoot PTR [edi]).facing ; esi = lastFacing
348
349     cmp ecx, 4
350     je SHOOT
351
352     mov ebx, (Dmoot PTR [edi]).drc.x
353     cmp ebx, 1
354     je RIGHT
355     cmp ebx, -1
356     je LEFT
357     ReturnX:
358
359     mov edx, (Dmoot PTR [edi]).drc.y
360     cmp edx, 1
361     je FALLING
362     cmp edx, -1
363     je JUMPING
364     cmp ecx, 2
365     je JUMPING
366     ReturnY:
367
368     cmp ebx, 0
369     je NoX
370     ReturnNoX:
371
372     cmp ecx, (Dmoot PTR [edi]).actionOn
373     jne ChangedAction
374
375     ;sameAction
376     cmp esi, (Dmoot PTR [edi]).facing
377     jne ChangedFacing
378     ret
379
380     RIGHT:
381     mov (Dmoot PTR [edi]).facing, 0
382     mov (Dmoot PTR [edi]).actionOn, 1
383     jmp ReturnX
384
385     LEFT:
386     mov (Dmoot PTR [edi]).facing, 1
387     mov (Dmoot PTR [edi]).actionOn, 1
388     jmp ReturnX
389
390     FALLING:
391     mov (Dmoot PTR [edi]).actionOn, 3
392     jmp ReturnY
```

```
393
394     JUMPING:
395     mov (Dmoot PTR [edi]).actionOn, 2
396     mov eax, PRE_JUMP
397     cmp (Dmoot PTR [edi]).jumpState, eax
398     je PreJump
399     ;invoke applyJump, offset zuko1
400     jmp ReturnY
401
402     PreJump:
403     ;;freezing
404     ret
405
406     NoX:
407     cmp edx,0
408     je Standing
409     jmp ReturnNoX
410
411     Standing:
412     mov (Dmoot PTR [edi]).actionOn,0
413     jmp ReturnNoX
414
415     SHOOT:
416     ret
417
418     ChangedAction:
419     invoke zukoResetAction, adrObj,cx;reset LastAction
420     ret
421
422     ChangedFacing:
423     invoke zukoResetAction,adrObj, ecx;reset LastAction
424     ret
425     zukoSetAction endp
426
427     moveFireBall proc adrObj: dword
428     mov ebx, adrObj
429     ASSUME ebx: PTR Dmoot
430     cmp [ebx].ball.isShowing, 0
431     je notShowing
432     ;;does show
433     cmp [ebx].ball.fireAnim.count, 0
434     jle stopFireBall
435
436     ;;move ball
437     mov edx, [ebx].ball.drcX
438     add [ebx].ball.pos.x,edx
439     ASSUME ebx: nothing
440     ret
441
442     notShowing:
443     ret
444
445     stopFireBall:
446     ASSUME ebx: PTR Dmoot
447     mov edx, [ebx].ball.fireAnim.ogCount
448     mov [ebx].ball.fireAnim.count, edx
```



```
449     mov [ebx].ball.isShowing,0
450     mov [ebx].ball.picIdx, 0
451     ASSUME ebx: nothing
452 ret
453 moveFireBall endp
454
455 handleZukoAdvance proc adrObj: dword
456     mov ebx,adrObj
457     mov edx,1
458     mov (Dmoot PTR [ebx]).advance, edx
459 ret
460 handleZukoAdvance endp
461 zukoMove proc adrObj: dword
462     mov eax, adrObj
463     ;;deal with freeze
464     mov edx, 2
465     cmp (Dmoot PTR [eax]).actionOn, edx
466     je JUMPING
467
468     RET_JUMPING:
469     invoke handleZukoAdvance, adrObj
470     mov eax,adrObj
471     mov ebx, (Dmoot PTR [eax]).advance
472     cmp ebx,0
473     je dontMove
474
475     ;;ELSE
476     invoke moveFireBall, adrObj
477     mov eax, adrObj
478     mov ebx, (Dmoot PTR [eax]).drc.x
479     add (Dmoot PTR [eax]).pos.x, ebx
480
481     mov edx, (Dmoot PTR [eax]).drc.y
482     add (Dmoot PTR [eax]).pos.y, edx
483     cmp edx,-1
484     je WHILEJUMPING
485     RET_WHILEJUMPING:
486 ret
487
488     JUMPING:
489     mov edx, PRE_JUMP
490     cmp (Dmoot PTR [eax]).jumpState, edx
491     je PREJUMP
492     jmp RET_JUMPING
493
494     PREJUMP:
495     ;;freeze
496
497 ret
498
499     WHILEJUMPING:
500     ;;jumpH--
501     mov edx, 1
502     sub (Dmoot PTR [eax]).jumpH, ebx
503 jmp RET_WHILEJUMPING
504
```

```
505     dontMove:
506     ret
507     zukoMove endp
508
509     fireBallScreenBounds proc adrObj:dword
510         mov ebx, adrObj
511         cmp (Dmoot PTR [ebx]).ball.isShowing, 0
512         je notShowing
513
514         ;;is showing
515         cmp (Dmoot PTR [ebx]).ball.pos.x, 0
516         jl TOO_LEFT
517
518         mov edx, 1200
519         sub edx, (Dmoot PTR [ebx]).ball.lastW
520         cmp (Dmoot PTR [ebx]).ball.pos.x, edx
521         jg TOO_RIGHT
522
523     notShowing:
524     ret
525
526     TOO_LEFT:
527     mov (Dmoot PTR [ebx]).ball.pos.x, 0
528     ret
529
530     TOO_RIGHT:
531     mov (Dmoot PTR [ebx]).ball.pos.x, edx
532     ret
533     fireBallScreenBounds endp
534     screenBounds proc adrObj: dword
535         mov ebx, adrObj
536         cmp (Dmoot PTR [ebx]).pos.x, 0
537         jl TOO_LEFT
538
539         mov edx, 1200
540         sub edx, 88
541         cmp (Dmoot PTR [ebx]).pos.x, edx;;minus zuko1 width
542         jg TOO_RIGHT
543
544     RET_X:
545
546         cmp (Dmoot PTR [ebx]).pos.y, 0
547         jl TOO_HIGH
548
549         mov edx, 700
550         sub edx, zukoH
551         cmp (Dmoot PTR [ebx]).pos.y, edx;;minus zuko1 width
552         jg TOO_LOW
553
554     RET_Y:
555     invoke fireBallScreenBounds, adrObj
556     ret
557
558     TOO_LEFT:
559     mov (Dmoot PTR [ebx]).pos.x, 0
560     jmp RET_X
```

```

561
562     TOO_RIGHT:
563     mov edx, 1200
564     sub edx, 88
565     mov (Dmoot PTR [ebx]).pos.x, edx
566     jmp RET_X
567
568
569     TOO_HIGH:
570     mov (Dmoot PTR [ebx]).pos.y, 0
571     jmp RET_Y
572
573     TOO_LOW:
574     mov edx, 700
575     sub edx, zukoH
576     mov (Dmoot PTR [ebx]).pos.y, edx
577     jmp RET_Y
578     screenBounds endp
579
580     mapCol proc adrObj: dword
581         ;;changes drcX acording to map objects
582         mov ebx, adrObj
583         mov eax, (Dmoot PTR [ebx]).drc.x
584
585         cmp eax,1
586         je RIGHT
587         cmp eax, -1
588         je LEFT
589     ret
590
591     RIGHT:
592     mov ecx, (Dmoot PTR [ebx]).adrRightPts
593     mov edx, (Dmoot PTR [ebx]).maxBytsRightPts
594     invoke checkPoints,adrObj, ecx,edx, groundColor, errorColor
595     cmp eax,1;;colors showing
596     je STOP
597     ret
598
599     LEFT:
600     mov ecx, (Dmoot PTR [ebx]).adrLeftPts
601     mov edx, (Dmoot PTR [ebx]).maxBytsLeftPts
602     invoke checkPoints,adrObj, ecx,edx, groundColor, errorColor
603     cmp eax,1;;colors showing
604     je STOP
605     ret
606
607     STOP:
608     mov ebx, adrObj
609     mov (Dmoot PTR [ebx]).drc.x,0
610     ret
611     mapCol endp
612
613     ; -----
614     gotHit proc adrObj:dword
615         mov eax, adrObj
616         mov ebx, ballDamage

```

```

617     sub (Dmoot PTR [eax]).HP, ebx
618     cmp (Dmoot PTR [eax]).HP, 0
619     jle DEATH
620 ret
621
622     DEATH:
623     mov (Dmoot PTR [eax]).HP, 0
624     mov ebx, (Dmoot PTR [eax]).lossCode
625     mov win, ebx
626 ret
627 gotHit endp
628 hit proc adrHit:dword, adrDef:dword
629     mov eax, adrHit
630     mov ebx, adrDef
631     cmp (Dmoot PTR [eax]).ball.isShowing, 1
632     jne NoHit
633
634     ;;check y hit
635     mov ecx, (Dmoot PTR [eax]).ball.pos.y;;ecx = yB
636     sub ecx, zukoH ;; ecx = yB-zukoH
637     mov edx, (Dmoot PTR [ebx]).pos.y ;;edx = yZ
638     cmp edx, ecx
639     jle NoHit
640     ;;if(yZ>=yB-zukoH) -> noHit
641
642     mov ecx, (Dmoot PTR [eax]).ball.pos.y;;ecx = yB
643     add ecx, ballH ;; ecx = yB + hB
644     cmp edx, ecx
645     jge NoHit
646     ;;if(yZ>=yB+hB) -> noHit
647
648     ;;HIT_Y
649     mov ecx, (Dmoot PTR [eax]).ball.pos.x;;ecx = xB
650     sub ecx, zukoW;; ecx = xB-Wz
651     mov edx, (Dmoot PTR [ebx]).pos.x ;;edx = xZ
652     cmp edx, ecx
653     jle NoHit
654     ;;if(xZ<=xB-Wz) 0> noHit
655
656     mov ecx, (Dmoot PTR [eax]).ball.pos.x;;ecx = xB
657     add ecx, (Dmoot PTR [eax]).ball.lastW ;; ecx = xB + wB
658     cmp edx, ecx
659     jge NoHit
660     ;;if(xZ>=xB+wB) -> noHit
661
662     jmp HIT
663     NoHit:
664 ret
665
666     HIT:
667     mov (Dmoot PTR [eax]).ball.fireAnim.count, 0
668     invoke playSoundAsync, offset hd.fireBall.hitSound
669     invoke gotHit, adrDef
670 ret
671 hit endp
672

```

```

673 ; -----
674
675 handleFireBallPicIdx proc adrObj:dword
676     mov ebx, adrObj
677     ASSUME ebx: PTR Dmoot
678     ;;count cannot be 0 here
679     ;;count--
680     mov edx, 1
681     sub [ebx].ball.fireAnim.count, edx
682     ;;if(count%cycle==0) picIdx++
683     mov eax, [ebx].ball.fireAnim.count;; eax = count
684     mov edi, [ebx].ball.fireAnim.cycle;;edi = cycle
685     mov edx,0
686     div edi; edi = eax%edi = count%cycle
687     cmp edx,0
688     je incPicIdx
689     ASSUME ebx: NOTHING
690     ret
691
692     incPicIdx:
693     ASSUME ebx: PTR Dmoot
694     mov edx,4
695     add [ebx].ball.picIdx, edx
696     ASSUME ebx: NOTHING
697     ret
698 handleFireBallPicIdx endp
699
700 drawFireBall proc adrObj:dword
701     mov ebx, adrObj
702     ASSUME ebx: PTR Dmoot
703     cmp [ebx].ball.isShowing,0
704     je notShowing
705
706     mov ecx, [ebx].ball.picIdx
707     mov edx,0
708     cmp [ebx].ball.drcX, edx
709     jg right
710     jl left
711
712     notShowing:
713     ASSUME ebx: nothing
714     ret
715
716     right:
717     mov ebx,adrObj
718     ASSUME ebx: PTR Dmoot
719     mov eax, [ebx].ball.pos.x
720     mov edx, [ebx].ball.pos.y
721     invoke drd_imageDrawCrop, offset ram.fireBall.moveR, eax,edx,
722         fireBallRSrcX[ecx],0,fireBallRW[ecx],ballH
723     mov ecx, [ebx].ball.picIdx
724     mov edx, fireBallRW[ecx]
725     mov [ebx].ball.lastW, edx
726     invoke handleFireBallPicIdx, adrObj
727     ASSUME ebx: nothing
727     ret

```

```

728
729     left:
730     ASSUME ebx: PTR Dmoot
731     invoke drd_imageDrawCrop, offset ram.fireBall.moveL, [ebx].ball.pos.x,
732         [ebx].ball.pos.y, fireBallLsrcX[ecx],0,fireBallLW[ecx],zukoH
733     mov ecx, [ebx].ball.picIdx
734     mov edx, fireBallRW[ecx]
735     mov [ebx].ball.lastW, edx
736     invoke handleFireBallPicIdx, adrObj
737     ASSUME ebx: nothing
738     ret
739 drawFireBall endp
740 displayDigit proc digit:dword, posX:dword, posY: dword
741
742     ;; idx = digit*4
743     mov eax, digit
744     mov edx,0
745     mov ecx, 4
746     mul ecx
747     ;;eax = idx
748     invoke drd_imageDrawCrop, offset res.digits, posX,posY, digitSrcX
749         [eax],0,digitW, digitH
750     ret
751 displayDigit endp
752 displayHP proc adrObj:dword
753     mov eax, adrObj
754     mov ebx, (Dmoot PTR [eax]).startPtHP.x
755     mov ecx, (Dmoot PTR [eax]).startPtHP.y
756     mov edx, (Dmoot PTR [eax]).HP
757
758     loopy:
759     mov edi,10
760     mov eax, edx
761     mov edx,0
762     div edi;; edx = eax%edi = hp%10
763     ;; eax = eax/edi
764     push eax
765     ;;digit, x, y
766     invoke displayDigit, edx, ebx, ecx
767     pop eax
768     cmp eax,0
769     je ENDLOOP
770     mov edx, eax ;;edx=hp
771     sub ebx, digitW
772     mov eax, adrObj
773     mov ecx, (Dmoot PTR [eax]).startPtHP.y
774     jmp loopy
775     ENDLOOP:
776     ret
777 displayHP endp
778
779 displayPlayerNum proc adrObj:dword
780     mov ebx, adrObj
781     mov eax, (Dmoot PTR [ebx]).playerNum

```

```

782     cmp eax,1
783     je PLAYER1
784
785     PLAYER2:
786     mov ecx, (Dmoot PTR [ebx]).pos.x
787     add ecx, (Dmoot PTR [ebx]).toPlayerNum.x
788     mov edx, (Dmoot PTR [ebx]).pos.y
789     add edx, (Dmoot PTR [ebx]).toPlayerNum.y
790
791     invoke drd_imageDraw, offset res.dig2, ecx,edx
792 ret
793
794     PLAYER1:
795     mov ecx, (Dmoot PTR [ebx]).pos.x
796     add ecx, (Dmoot PTR [ebx]).toPlayerNum.x
797     mov edx, (Dmoot PTR [ebx]).pos.y
798     add edx, (Dmoot PTR [ebx]).toPlayerNum.y
799
800     invoke drd_imageDraw, offset res.dig1, ecx,edx
801 ret
802 displayPlayerNum endp
803
804 handleZukoPicIdx proc adrObj:dword, toOgCountAdr: dword, toCountAdr: dword,  ↗
805     toCycleAdr: dword
806     mov ebx, adrObj
807     add ebx, toCountAdr
808     mov eax, [ebx]
809     cmp eax,0;;if count==0 reset picIdx and count
810     je resetCountAndPicIdx
811
812     ;;count--
813     mov eax, 1
814     sub [ebx],eax
815
816     ;;if(count%cycle==0) picIdx++
817     mov eax, [ebx];;eax = count
818     mov ebx,adrObj
819     add ebx, toCycleAdr
820     mov edi, [ebx];;edi = cycle
821     mov edx, 0
822     div edi;;edx = eax%edi = count%cycle
823     cmp edx,0
824     je incPicIdx
825
826     incPicIdx:
827     mov ebx,4;;picIdx++
828     mov eax, adrObj
829     add (Dmoot PTR [eax]).picIdx, ebx
830 ret
831
832     resetPicIdx:
833     mov eax, adrObj
834     mov (Dmoot PTR [eax]).picIdx, 0
835 ret
836

```

```

837     resetCountAndPicIdx:
838     mov  eax, adrObj
839     mov  (Dmoot PTR [eax]).picIdx, 0
840
841     ;reset count by ogCount
842     ASSUME  eax: nothing
843     mov  ebx, adrObj
844     add  ebx, toOgCountAdr
845     mov  eax, adrObj
846     add  eax, toCountAdr
847     mov  edx, [ebx]
848     mov  [eax], edx
849     ret
850 handleZukoPicIdx  endp
851
852 zukoDrawPic  proc  ramImgAdr:dword, posX:dword, posY:dword, srcX:dword,      ↗
                    widZZ:dword, adrObj:dword, toOgCount:dword, toCount:dword, toCycle:dword
853     ASSUME  eax: nothing
854     invoke drd_imageDrawCrop, ramImgAdr, posX, posY, srcX, 0, widZZ, zukoH
855     invoke handleZukoPicIdx, adrObj, toOgCount, toCount, toCycle
856     ret
857 zukoDrawPic  endp
858
859 drawZuko  proc  adrObj:dword
860     invoke drawFireBall, adrObj
861     invoke displayHP, adrObj
862     invoke displayPlayerNum, adrObj
863
864     mov  eax, adrObj
865     ASSUME  eax:PTR Dmoot
866
867     mov  ecx, [eax].picIdx;ecx = picIdx
868     mov  edx, [eax].pos.x
869     mov  edi, [eax].pos.y
870
871     cmp  [eax].actionOn, 0
872     je  standState
873     cmp  [eax].actionOn, 3
874     je  fallState
875     cmp  [eax].actionOn, 2;jumping
876     je  Jump
877     cmp  [eax].drc.x, 1
878     je  runRState
879     cmp  [eax].drc.x, -1
880     je  runLState
881     cmp  [eax].actionOn, 4
882     je  shoot
883     ret
884
885     standState:
886     cmp  [eax].facing, 0
887     je  standR
888     ;;left
889     invoke zukoDrawPic, offset ram.zuko.standL, edx, edi, zukoStandLsrcX      ↗
                    [ecx], zukoStandWid, adrObj, Dmoot.standAnim.ogCount,      ↗
                    Dmoot.standAnim.count, Dmoot.standAnim.cycle

```



```
890 ret
891
892 standR:
893 invoke zukoDrawPic,offset ram.zuko.standR,edx,edi,zukoStandRsrcX
    [ecx],zukoStandWid, adrObj, Dmoot.standAnim.ogCount,
    Dmoot.standAnim.count, Dmoot.standAnim.cycle
894 ret
895
896 runRState:
897 invoke zukoDrawPic,offset ram.zuko.runR,edx,edi,zukoRunRsrcX[ecx],
    zukoRunRW[ecx],adrObj, Dmoot.runAnim.ogCount, Dmoot.runAnim.count,
    Dmoot.runAnim.cycle
898 ret
899
900 runLState:
901 invoke zukoDrawPic,offset ram.zuko.runL,edx,edi,zukoRunLsrcX[ecx],
    zukoRunLW[ecx],adrObj, Dmoot.runAnim.ogCount, Dmoot.runAnim.count,
    Dmoot.runAnim.cycle
902 ret
903
904 fallState:
905 cmp [eax].facing,0
906 je fallR
907 ;;left
908 invoke zukoDrawPic,offset ram.zuko.fallL,edx,edi,zukoFallLsrcX[ecx],
    zukoFallLW[ecx],adrObj, Dmoot.fallAnim.ogCount, Dmoot.fallAnim.count,
    Dmoot.fallAnim.cycle
909 ret
910
911 fallR:
912 invoke zukoDrawPic,offset ram.zuko.fallR,edx,edi,zukoFallRsrcX[ecx],
    zukoFallRW[ecx],adrObj, Dmoot.fallAnim.ogCount, Dmoot.fallAnim.count,
    Dmoot.fallAnim.cycle
913 ret
914
915 Jump:
916 cmp [eax].facing,0
917 je jumpR
918 ;;left
919 ;;split pre-while
920 mov esi, PRE_JUMP
921 cmp [eax].jumpState, esi
922 je preJumpL
923 ;;jumpL
924 invoke zukoDrawPic,offset ram.zuko.jumpL,edx,edi,zukoJumpLsrcX[ecx],
    zukoJumpLW[ecx],adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
    Dmoot.jumpAnim.cycle
925 ret
926
927 preJumpL:
928 invoke zukoDrawPic,offset ram.zuko.jumpL,edx,edi,zukoPreJumpLsrcX,
    zukoPreJumpLW,adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
    Dmoot.jumpAnim.cycle
929 ret
930
931 jumpR:
932 mov esi, PRE_JUMP
933 cmp [eax].jumpState, esi
934 je preJumpR
```

```

932     invoke zukoDrawPic,offset ram.zuko.jumpR,edx,edi,zukoJumpRSrcX[ecx],
           zukoJumpRW[ecx],adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
           Dmoot.jumpAnim.cycle
933 ret
934
935     preJumpR:
936     invoke zukoDrawPic,offset ram.zuko.jumpR,edx,edi,zukoPreJumpRSrcX,
           zukoPreJumpRW,adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
           Dmoot.jumpAnim.cycle
937 ret
938
939     shoot:
940     cmp [eax].facing, 0
941     je shootR
942     ;;shoot L
943     invoke zukoDrawPic,offset ram.zuko.shootL,edx,edi,zukoShootLSrcX[ecx],
           zukoShootLW[ecx],adrObj, Dmoot.shootAnim.ogCount,
           Dmoot.shootAnim.count, Dmoot.shootAnim.cycle
944 ret
945
946     shootR:
947     invoke zukoDrawPic,offset ram.zuko.shootR,edx,edi,zukoShootRSrcX[ecx],
           zukoShootRW[ecx],adrObj, Dmoot.shootAnim.ogCount,
           Dmoot.shootAnim.count, Dmoot.shootAnim.cycle
948 ret
949 drawZuko endp
950
951 ;-----
952 winScreen proc adrObj1: dword, adrObj2: dword
953     mov eax, win
954     mov ebx, adrObj1
955     cmp eax, (Dmoot PTR [ebx]).playerNum
956     je PLAYER1WON
957
958     PLAYER2WON:
959     invoke drd_imageDraw,offset res.win2,0,0
960
961 ret
962     PLAYER1WON:
963     invoke drd_imageDraw,offset res.win1,0,0
964
965 ret
966 winScreen endp
967
968 resetGame proc adrObj: dword
969     mov eax, adrObj
970     mov ebx, (Dmoot PTR [eax]).startPos.x
971     mov (Dmoot PTR [eax]).pos.x, ebx
972
973     mov ebx, (Dmoot PTR [eax]).startPos.y
974     mov (Dmoot PTR [eax]).pos.y, ebx
975
976     mov ebx, StartHP
977     mov (Dmoot PTR [eax]).HP, ebx
978
979     mov ebx, (Dmoot PTR [eax]).actionOn

```

```
980     invoke zukoResetAction, adrObj, ebx
981 ret
982 resetGame endp
```

```

1  include \masm32\include\user32.inc;;sound
2  includelib \masm32\lib\user32.lib
3
4  include \masm32\include\winmm.inc;;sound
5  includelib \masm32\lib\winmm.lib
6
7  .code
8  zukoDflt proc adrObj:dword
9      mov ebx, adrObj
10     ASSUME ebx: PTR Dmoot
11     mov [ebx].drc.x, 0
12     mov [ebx].drc.y, 0
13     ASSUME ebx: nothing
14 ret
15 zukoDflt endp
16
17 zukoActByKey proc adrObj: dword ,toKeyAdr: dword, toDrcAdr:dword, valuu:dword
18     ;;if pressed insert value
19     mov ebx, adrObj
20     add ebx, toKeyAdr
21     invoke GetAsyncKeyState, [ebx]
22     cmp eax, 0
23     jne PRESSED;jne
24 ret
25
26     PRESSED:
27     mov edx, valuu
28     mov edi, adrObj
29     add edi, toDrcAdr
30     mov [edi], edx
31 ret
32 zukoActByKey endp
33
34 checkPoints proc adrObj:dword, adrArr:dword, sizeofArr:dword, color1:dword,  ↗
    color2: dword
35 ;                                     offset of arr[0], max bytes in arr , bgr, bgr
36
37 ;result in eax -> 0-colors don't show, 1- one or both colors show
38
39     invoke drd_beginHdc;get handle
40     mov esi, eax ; esi = handle
41     mov ecx, 0;ecx = idx
42     dloop:
43         mov ebx, adrObj
44
45         mov edi, (Dmoot PTR [ebx]).pos.x
46         mov edx, (Dmoot PTR [ebx]).pos.y
47
48         push ecx
49         push esi
50
51         mov esi, ecx
52         add esi, adrArr;esi=adrArr+ idx
53         add edi, [esi];point[idx].x
54         add edx, [esi + 4];point[idx].y
55         ;; edi = ctrlPt1X, edx = ctrlPt1Y

```

```
56     pop esi;hdc = esi
57     mov eax, esi ;hdc = eax
58     invoke GetPixel, eax, edi,edx ; eax = pixel
59     cmp eax, color1
60     je SHOW
61     cmp eax, color2
62     je SHOW
63
64     pop ecx; ecx = idx
65     add ecx,8
66     cmp ecx, sizeofArr
67     jl dloop;if(ecx+8<sizeofArr) stay in loop
68
69     NOTSHOW:
70     mov eax,0
71     invoke drd_endHdc, esi
72     ret
73
74     SHOW:
75     invoke drd_endHdc, esi
76     mov eax,1
77     ret
78     checkPoints endp
79
80     playSoundAsync proc adrPath:dword
81         invoke PlaySound , adrPath,NULL, SND_ASYNC
82     ret
83     playSoundAsync endp
```