The Fire Ball Game

פרויקט בשפת אסמבלי

אלון דיין י'5

מוגש ליהודה אור, הכפר הירוק

יוני 2019

תוכן עניינים

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.

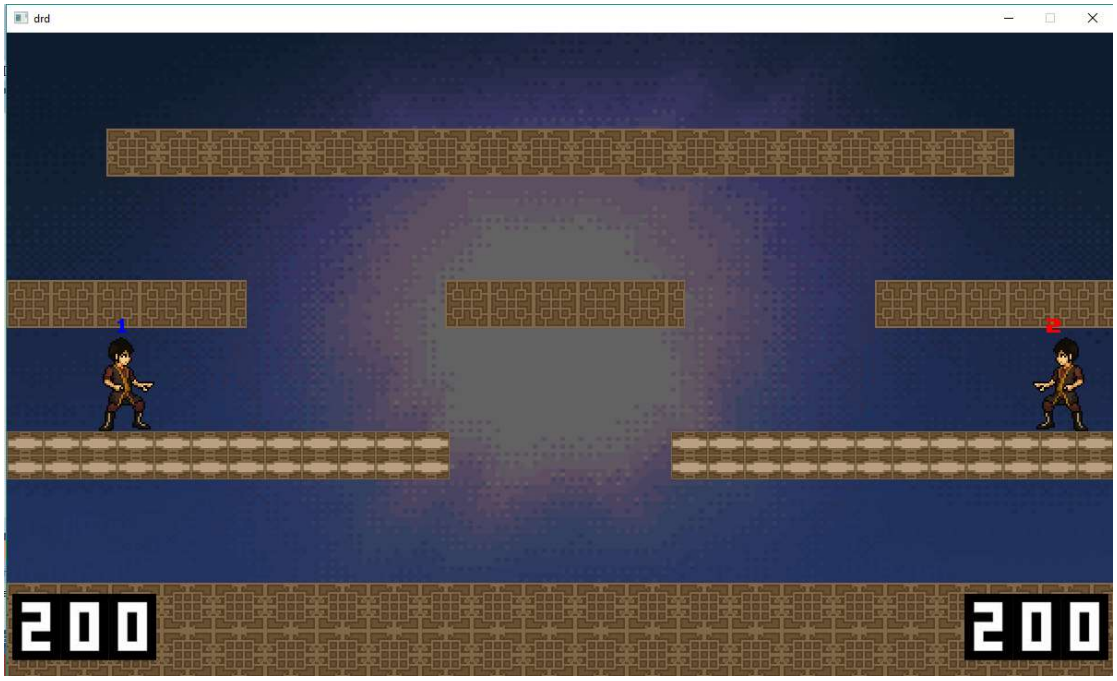
בזמן הפרויקט, שיפרתי את הדרך בה אני חושב על כתיבת קוד. עד עתה, כאשר הייתי צריך לכתוב קוד בשפה עילית הייתי חושב על התוכנית הכללית שלי ומתחיל לכתוב אותו מההתחלה לסוף. לעומת זאת, כאשר רציתי לכתוב קוד באסמבלי, גיליתי ששיטה זו אינה יעילה ובאמצע כתיבת הקוד גיליתי טעויות ב"תכנית" ההתחלתית שלי. לכן, די מהר בתחילת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

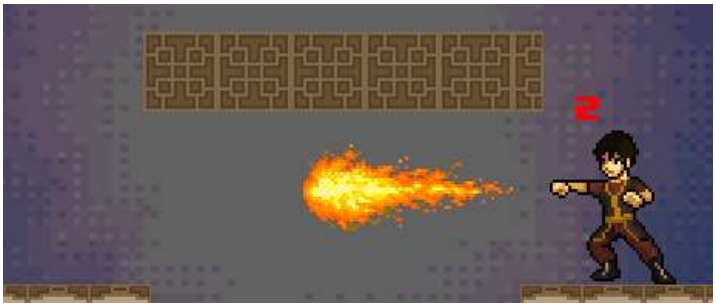
המשחק שלי

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



שחקן מספר 1 בעת קפיצה.

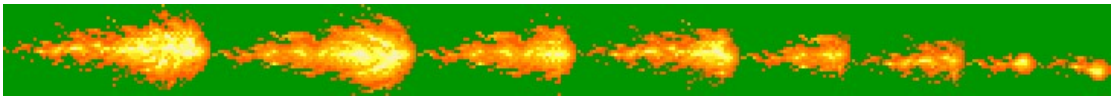
שחקן מספר 2 בעת ירייה.



שחקן מספר 2 בעת ריצה

מסך הניצחון לשחקן 1.

רצף התמונות לאנימציית 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.

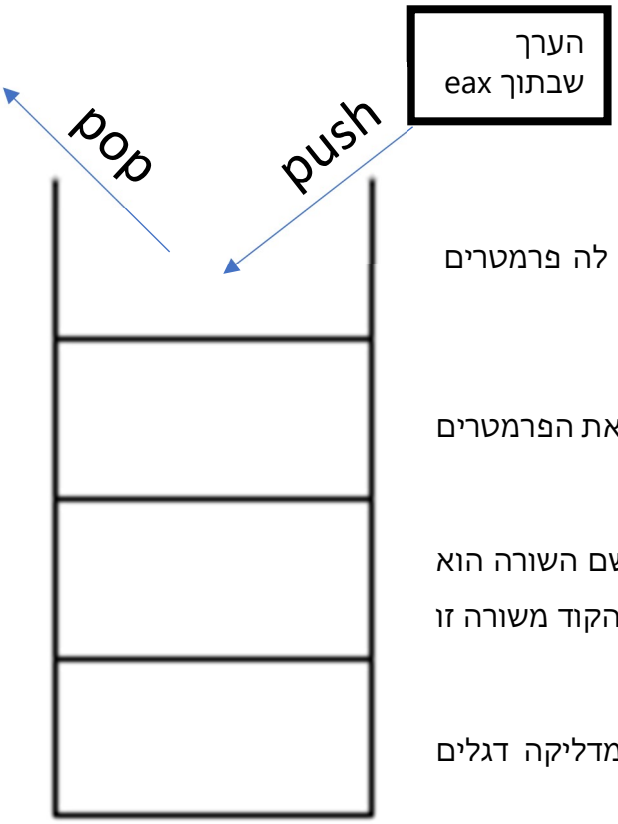
sub – פקודה המקטינה אוגר/גודל בזכרון בגודל אחר. לדוגמה: `sub var,eax` = הקטן את המשתנה var בערך של eax.

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 edx:eax בפרמטר המתקבל .התוצאה נשמרת ב-eax והשארית נשמרת ב-edx. לדוגמה אם eax=4, ebx=7 וגם edx=0 אז הפקודה `div ebx` תגרום לכך: eax = 1, edx = 3.

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 ביניהם.

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.

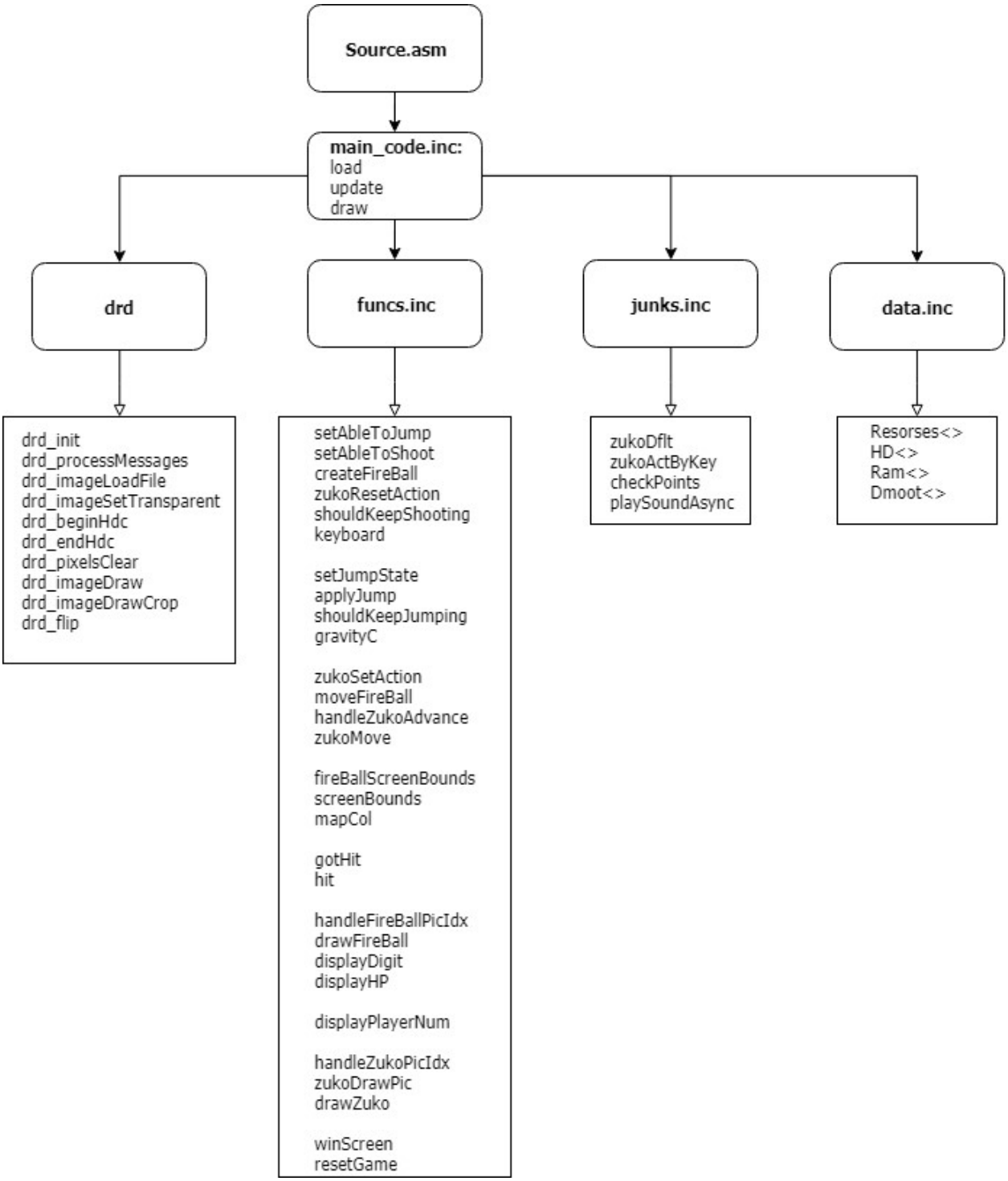
.data – שורה זו מצהירה כי משורה זו והלאה, זהו החלק שבו נצהיר על משתנים. משתנים יכולים להיות מסוג byte, word ו-dword או מבני נתונים מורכבים יותר כגון מערך או struct אחר שהמתכנת יוצר.

[] – 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 `offset HP` 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-<>Img 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 לאנימציית 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 ויריה).

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 הנתיבים שלהם (path).

HD<> - מכיל את הנתיבים 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

```
checkPoints proc adrObj:dword, adrArr:dword,sizeofArr:dword, color1:dword,
  color2: dword
;                                offset of arr[0], max bytes in arr , bgr, bgr

;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 - `0x00bbggrr`).

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
```

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 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.

כעת, נתחיל את הלולאה.

```
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
    jl dloop;if(ecx+8<sizeofArr) stay in loop
```

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.

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.

```
    NOTSHOW:
    mov eax,0
    invoke drd_endHdc, esi
ret

    SHOW:
    invoke drd_endHdc, esi
    mov eax,1
ret
```

הפונקציה בשלמותה:

```
checkPoints proc adrObj:dword, adrArr:dword,sizeofArr:dword, color1:dword,
  color2: dword
;                                offset of arr[0], max bytes in arr , bgr, bgr

;result in eax -> 0-colors don't show, 1- one or both colors show

    invoke drd_beginHdc;get handle
    mov esi, eax ; esi = handle
    mov ecx,0;ecx = idx
    dloop:
        mov ebx, adrObj

        mov edi, (Dmoot PTR [ebx]).pos.x
        ;mov edi, [ebx+DM.pos.x]
        mov edx, [ebx+DM.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
        jl dloop;if(ecx+8<sizeofArr) stay in loop

    NOTSHOW:
    mov eax,0
    invoke drd_endHdc, esi
ret

    SHOW:
    invoke drd_endHdc, esi
    mov eax,1
ret
checkPoints endp
```

```
include \masm32\include\masm32rt.inc
include main_code.inc
.code

main proc
    invoke load
    loopi:
        invoke update
        invoke draw
    jmp loopi
ret
main endp

end main
```

```
include drd.inc
includelib drd.lib
include data.inc
include junks.inc
include funcs.inc
.code

load proc
    invoke drd_init,1200,700,0;;open window
    ;;load images
    invoke drd_imageLoadFile,offset res.bgpath1,offset res.bg1
    invoke drd_imageLoadFile,offset res.digitsPath,offset res.digits

    invoke drd_imageLoadFile,offset res.win1Path,offset res.win1
    invoke drd_imageLoadFile,offset res.win2Path,offset res.win2

    invoke drd_imageLoadFile,offset res.dig1Path,offset res.dig1
    invoke drd_imageSetTransparent,offset res.dig1,0ffffffh

    invoke drd_imageLoadFile,offset res.dig2Path,offset res.dig2
    invoke drd_imageSetTransparent,offset res.dig2,0ffffffh

    invoke drd_imageLoadFile, offset hd.zuko.standR, offset ram.zuko.standR
    invoke drd_imageSetTransparent,offset ram.zuko.standR,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.standL, offset ram.zuko.standL
    invoke drd_imageSetTransparent,offset ram.zuko.standL,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.runR, offset ram.zuko.runR
    invoke drd_imageSetTransparent,offset ram.zuko.runR,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.runL, offset ram.zuko.runL
    invoke drd_imageSetTransparent,offset ram.zuko.runL,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.fallR, offset ram.zuko.fallR
    invoke drd_imageSetTransparent,offset ram.zuko.fallR,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.fallL, offset ram.zuko.fallL
    invoke drd_imageSetTransparent,offset ram.zuko.fallL,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.jumpR, offset ram.zuko.jumpR
    invoke drd_imageSetTransparent,offset ram.zuko.jumpR,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.jumpL, offset ram.zuko.jumpL
    invoke drd_imageSetTransparent,offset ram.zuko.jumpL,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.shootR, offset ram.zuko.shootR
    invoke drd_imageSetTransparent,offset ram.zuko.shootR,0009600h

    invoke drd_imageLoadFile, offset hd.zuko.shootL, offset ram.zuko.shootL
    invoke drd_imageSetTransparent,offset ram.zuko.shootL,0009600h

    invoke drd_imageLoadFile, offset hd.fireBall.moveR, offset ram.fireBall.moveR
    invoke drd_imageSetTransparent,offset ram.fireBall.moveR,0009600h

```

```
    invoke drd_imageLoadFile, offset hd.fireBall.moveL, offset ram.fireBall.moveL
    invoke drd_imageSetTransparent,offset ram.fireBall.moveL,0009600h
ret
load endp

update proc
    invoke drd_processMessages;;allow window to be closed

    ;;check if someone won the game
    cmp win, 0
    jne SOMEONEWON

    invoke zukoDflt, offset zuko1
    invoke zukoDflt, offset zuko2

    invoke keyboard, offset zuko1
    invoke keyboard, offset zuko2

    invoke gravityC, offset zuko1
    invoke gravityC, offset zuko2

    invoke zukoSetAction, offset zuko1
    invoke zukoSetAction, offset zuko2

    invoke mapCol, offset zuko1
    invoke mapCol, offset zuko2

    invoke zukoMove, offset zuko1
    invoke zukoMove, offset zuko2

    invoke screenBounds, offset zuko1
    invoke screenBounds, offset zuko2

    invoke hit, offset zuko1, offset zuko2
    invoke hit, offset zuko2, offset zuko1
ret

    SOMEONEWON:
    ;;skip update
    invoke zukoActByKey,0,offset Rkey,offset win, 0
    cmp win,0
    je RESET
ret

    RESET:
    invoke resetGame, offset zuko1
    invoke resetGame, offset zuko2
ret
update endp

draw proc
        invoke drd_pixelsClear,0;;clear screen

        ;;check if someone won
        cmp win,0
```

```
        jne SOMEONEWON

        invoke drd_imageDraw,offset res.bg1,0,0

        invoke drawZuko, offset zuko1
        invoke drawZuko, offset zuko2

        invoke drd_flip
ret

    SOMEONEWON:
    invoke winScreen, offset zuko1, offset zuko2
    invoke drd_flip
ret
draw endp
```

```
.const
;;structs
HDzuko struct
    standR byte "zuko/standR.png",0
    standL byte "zuko/standL.png",0
    runR byte "zuko/runR.png",0
    runL byte "zuko/runL.png",0
    jump byte "zuko/jump.png",0
    fallR byte "zuko/fallR.png",0
    fallL byte "zuko/fallL.png",0
    jumpR byte "zuko/jumpR.png",0
    jumpL byte "zuko/jumpL.png",0
    shootR byte "zuko/shootR.png",0
    shootL byte "zuko/shootL.png",0
HDzuko ends

HDfireBall struct
    moveR byte "otherPics/fireBallR.png",0
    moveL byte "otherPics/fireBallL.png",0
    fireSound byte "sound/fireBall.wav",0
    hitSound byte "sound/hit.wav",0
HDfireBall ends

HD struct
    zuko HDzuko<>
    fireBall HDfireBall<>
HD ends

AnimfireBall struct
    moveR Img<>
    moveL Img<>
AnimfireBall ends

Anim struct
    standR Img<>
    standL Img<>
    runR Img<>
    runL Img<>
    fallR Img<>
    fallL Img<>
    jumpR Img<>
    jumpL Img<>
    shootR Img<>
    shootL Img<>
Anim ends

Ram struct
    zuko Anim<>
    fireBall AnimfireBall<>
Ram ends

Resorses struct
    bgpath1 byte "otherPics/newFullBg.png",0
    bg1 Img<>
    digitsPath byte "otherPics/digits.png",0
    digits Img<>
```

```
    win1Path byte "otherPics/player1won.png",0
    win1 Img<>
    win2Path byte "otherPics/player2won.png",0
    win2 Img<>
    dig1Path byte "otherPics/dig1.png",0
    dig1 Img<>
    dig2Path byte "otherPics/dig2.png",0
    dig2 Img<>
Resorses ends

Keys struct
    left dword ?
    right dword ?
    up dword ?
    shoot dword ?
Keys ends

Action struct
    ogCount dword ?
    count dword ?
    cycle dword ?
    ;ogCount/cycle = number of frames in sprite sheet
    ;ogCount = cycle*number_of_frames_in_sprite_sheet
Action ends

fireBall struct
    pos POINT<0,0>
    drcX dword 0
    isShowing dword 0; 0- not showing, 1- showing
    fireAnim Action<400,400,50>
    picIdx dword 0;;idx in Anim
    lastW dword 0
fireBall ends

Dmoot struct
    pos POINT<>
    startPos POINT<>;;the position Dmoot starts at the begining of the game
    drc POINT<>
    keys Keys<>
    adrUpPts dword ?
    maxBytsUpPts dword ?
    adrLeftPts dword ?
    maxBytsLeftPts dword ?
    adrRightPts dword ?
    maxBytsRightPts dword ?

    startPtHP POINT<>;; the place to draw the HP at
    lossCode dword ?
    playerNum dword ?

    toPlayerNum POINT <15,-22>

    moveAct Action<4,4,2>
    standAnim Action<100,100,50>
    runAnim Action<200,200,25>
    shootAnim Action<200,200,50>
```

```
    fallAnim Action<100,100,50>
    jumpAnim Action<80,80,25>


    picIdx dword 0;;idx in animation
    facing dword 0; 0-right, 1-left
    actionOn dword 0 ;; 0-stand, 1-run, 2-jump, 3-fall, 4-shooting
    jumpState dword 0 ;; 0-pre, 1-jump
    advance dword 1 ;; 0-don't, 1-do

    ogJumpH dword 204
    jumpH dword 204 ;; at that height starts to fall after jump
    ogJumpFreeze dword 50
    jumpFreeze dword 50;;freeze before jump for that amount of time

    ctrlPt1 POINT<0,102>;fall
    ctrlPt2 POINT<76,102>;fall

    ball fireBall<>
    HP dword 200
Dmoot ends

.data

emptyDword dword 0
Rkey dword VK_R

res Resorses<>
hd HD<>
ram Ram<>

win dword 0 ;; 0-game, 2-zuko2 won, 1-zuko1 won

StartHP dword 200

UpPts POINT 2 dup(<0,-1>,<66,-1>)
LeftPts POINT 3 dup(<-1,0>,<-1,52>,<-1,101>)
RightPts POINT 6 dup(<76,0>,<76,52>,<76,101>, <66,0>,<66,52>,<66,101>)

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
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

groundColor dword 000486880h
errorColor dword 000ffffffh
blackColor dword 000000000h

zukoH dword 102
zukoW dword 76

zukoStandWid dword 62
zukoStandRSrcX dword 2 dup(0,78)
zukoStandLSrcX dword 2 dup(78,0)
```

```
zukoRunRW dword 8 dup(77,88,82,64,77,86,82,66)
zukoRunRSrcX dword 8 dup(0,78,167,250,315,393,480,563)
zukoRunLW dword 8 dup(66,82,86,77,64,82,88,77)
zukoRunLSrcX dword 8 dup(1,68,151,238,316,381,464,553)

zukoShootRW dword 4 dup(64,54,82,82)
zukoShootRSrcX dword 4 dup(0,65,120,203)
zukoShootLW dword 4 dup(64,54,82,82)
zukoShootLSrcX dword 4 dup(222,167,84,1)

zukoFallRW dword 4 dup(76,76,66,66)
zukoFallRSrcX dword 4 dup(0,77,154,221)
zukoFallLW dword 4 dup(66,66,76,76)
zukoFallLSrcX dword 4 dup(1,67,134,211)

zukoPreJumpRW dword 58
zukoPreJumpRSrcX dword 1
zukoJumpRW dword 8 dup(78,78,68,68,68,68,78,78)
zukoJumpRSrcX dword 8 dup(59,138,217,286,286,217,138,59)
zukoPostJumpRW dword 3 dup(58,65,58)
zukoPostJumpRSrcX dword 3 dup(355,414,480)

zukoPreJumpLW dword 58
zukoPreJumpLSrcX dword 481
zukoJumpLW dword 8 dup(78,78,68,68,68,68,78,78)
zukoJumpLSrcX dword 8 dup(402,323,254,185,185,254,323,402)
zukoPostJumpLW dword 3 dup(58,65,58)
zukoPostJumpLSrcX dword 3 dup(126,60,59)

;flags for jump State
PRE_JUMP dword 0
WHILE_JUMP dword 1

START_FIREBALL dword 50

ctrlPts POINT 2 dup(<0,102>,<56,102>)

createBallR POINT <83,11>
createBallL POINT <-136,11>

ballH dword 60
fireBallRW dword 8 dup(136,134,104,106,72,74,46,48)
fireBallRSrcX dword 8 dup(0,137,272,377,484,557,632,679)
fireBallLW dword 8 dup(136,134,104,106,72,74,46,48)
fireBallLSrcX dword 8 dup(591,456,351,244,171,96,49,0)
ballDamage dword 50

digitH dword 72
digitW dword 52
digitSrcX dword 10 dup(0,52,104,156,208,260,312,364,416,468)
```

```
.code

setAbleToJump proc adrObj: dword
;;result in eax: 1-can jump, 0- cannot jump
;; cannot jump if falling or jumping
    mov ebx, adrObj
    mov ecx, (Dmoot PTR [ebx]).actionOn
    cmp ecx, 3 ;fall
    je CANNOT
    cmp ecx, 2 ;;jump
    je CANNOT

    ;can jump
    mov eax,1
ret

    CANNOT:
    mov eax,0
ret

setAbleToJump endp

setAbleToShoot proc adrObj:dword
;;result in eax: 1-can jump, 0- cannot jump
    mov ebx, adrObj
    mov ecx, (Dmoot PTR [ebx]).actionOn
    cmp ecx, 3 ;fall
    je CANNOT
    cmp ecx, 2 ;;jump
    je CANNOT
    cmp ecx, 4;shooting
    je CANNOTSH

    CAN:
    mov eax, 1
ret

    CANNOT:
    mov eax,0
ret

CANNOTSH:
    mov eax,0
ret
setAbleToShoot endp

createFireBall proc adrObj:dword
    invoke playSoundAsync, offset hd.fireBall.fireSound

    ;zuko1.ball = fireBall
    mov ebx, adrObj
    ASSUME ebx: PTR Dmoot
    mov edx, [ebx].ball.fireAnim.ogCount
    mov [ebx].ball.fireAnim.count, edx
    mov [ebx].ball.picIdx, 0

```

```
    mov eax, [ebx].pos.x
    mov ecx, [ebx].pos.y

    mov edx, [ebx].facing
    cmp edx,1
    je LeftPt
    cmp edx,0
    je RightPt

    retPt:
    mov [ebx].ball.pos.x, eax
    mov [ebx].ball.pos.y, ecx

    mov edx, [ebx].facing
    cmp edx,1
    je Left
    cmp edx,0
    je Right

    RET_DRC:
    mov [ebx].ball.drcX, edx

    mov edx,1
    mov [ebx].ball.isShowing, edx
    ASSUME ebx: nothing
ret

    RightPt:
    add eax, createBallR.x
    add ecx, createBallR.y
jmp retPt

    LeftPt:
    add eax, createBallL.x
    add ecx, createBallL.y
jmp retPt

    Right:
    mov edx,2
jmp RET_DRC

    Left:
    mov edx,-2
jmp RET_DRC
createFireBall endp

zukoResetAction proc adrObj: dword, act: dword
    mov eax, adrObj
    ASSUME eax: PTR Dmoot
    mov [eax].picIdx, 0;reset PicIdx
    mov ebx,act

    cmp ebx, 0
    je ResetStand
    cmp ebx,1
    je ResetRunAnim
```

```
    cmp ebx, 3
    je ResetFallAnim
    cmp ebx,2
    je ResetJumpAnim
    cmp ebx, 4
    je ResetShootAnim

    RETURN:
    ASSUME eax: nothing
ret

    ResetStand:
    ASSUME eax: PTR Dmoot
    mov edx, [eax].standAnim.ogCount
    mov [eax].standAnim.count, edx
jmp RETURN

    ResetRunAnim:
    ASSUME eax: PTR Dmoot
    mov edx, [eax].runAnim.ogCount
    mov [eax].runAnim.count, edx
jmp RETURN

    ResetFallAnim:
    ASSUME eax: PTR Dmoot
    mov edx, [eax].fallAnim.ogCount
    mov [eax].fallAnim.count,edx
jmp RETURN

    ResetJumpAnim:
    ASSUME eax: PTR Dmoot
    mov edx, [eax].jumpAnim.ogCount
    mov [eax].jumpAnim.count, edx
jmp RETURN

    ResetShootAnim:
    ASSUME eax: PTR Dmoot
    mov edx, [eax].shootAnim.ogCount
    mov [eax].shootAnim.count, edx
jmp RETURN
zukoResetAction endp

shouldKeepShooting proc adrObj:dword
    mov ebx,adrObj
    mov edx, START_FIREBALL
    cmp (Dmoot PTR [ebx]).shootAnim.count, edx
    je CREATEFIREBALL
    cmp (Dmoot PTR [ebx]).shootAnim.count, 0
    je STOP_SHOOTING
ret

    STOP_SHOOTING:
    invoke zukoResetAction,adrObj, 4
    mov ebx, adrObj
    mov (Dmoot PTR [ebx]).actionOn,0
ret
```

```

    CREATEFIREBALL:
    invoke createFireBall, adrObj
ret
shouldKeepShooting endp

keyboard proc adrObj: dword
    mov ebx, adrObj

    mov edx, 4
    cmp (Dmoot PTR [ebx]).actionOn, edx
    je WHILE_SHOOT

    invoke zukoActByKey,adrObj,Dmoot.keys.left,Dmoot.drc.x, -1
    invoke zukoActByKey,adrObj,Dmoot.keys.right,Dmoot.drc.x, 1

    invoke setAbleToJump, adrObj
    cmp eax,0
    je CANNOTJUMP

    ;;can jump
    invoke zukoActByKey, adrObj, Dmoot.keys.up,Dmoot.drc.y, -1

    CANNOTJUMP:
    ;;empty jump
    mov edx,ebx
    add edx, Dmoot.keys.up
    invoke zukoActByKey, 0,edx, offset emptyDword, 0

    invoke setAbleToShoot, adrObj
    cmp eax,0
    je CANNOTSHOOT

    ;;can shoot
    invoke zukoActByKey, adrObj,Dmoot.keys.shoot,Dmoot.actionOn, 4
    mov edx, 4
    cmp (Dmoot PTR [ebx]).actionOn, edx
    je FIRST_SHOOT

    CANNOTSHOOT:
    ;;empty shoot
    mov edx,ebx
    add edx, Dmoot.keys.shoot
    invoke zukoActByKey, 0,edx, offset emptyDword, 0
ret

    FIRST_SHOOT:
ret

    WHILE_SHOOT:
    invoke shouldKeepShooting, ebx
ret
keyboard endp

setJumpState proc adrObj: dword, jmpStt: dword
    mov ebx, adrObj
```

```
    mov ecx, jmpStt
    mov (Dmoot PTR [ebx]).jumpState, ecx
ret
setJumpState endp

applyJump proc adrObj: dword
    mov ebx, adrObj
    mov edx, PRE_JUMP
    cmp (Dmoot PTR [ebx]).jumpState, edx
    je FREEZE
    NO_MORE_FREEZE:
    ;;change drc
    mov (Dmoot PTR [ebx]).drc.y, -1
ret

    FREEZE:
    cmp (Dmoot PTR [ebx]).jumpFreeze, 0
    je PastFreeze
    ;;jumpFreeze--
    mov edx,1
    sub (Dmoot PTR [ebx]).jumpFreeze, edx
ret

    PastFreeze:
    ;reset jump freeze
    mov edx, (Dmoot PTR [ebx]).ogJumpFreeze
    mov (Dmoot PTR [ebx]).jumpFreeze, edx
    ;change jumpState
    mov edx, WHILE_JUMP
    mov (Dmoot PTR [ebx]).jumpState, edx
    jmp NO_MORE_FREEZE
applyJump endp

shouldKeepJumping proc adrObj:dword
;; result in actionOn
    mov ecx, adrObj

    mov ebx, (Dmoot PTR [ecx]).adrUpPts
    mov esi, (Dmoot PTR [ecx]).maxBytsUpPts
    invoke checkPoints, adrObj,ebx,esi, groundColor, errorColor

    mov ecx,adrObj
    ASSUME ecx: PTR Dmoot
    cmp eax,1;colors showing
    je StopJump

    ;;check jumpH
    cmp [ecx].jumpH, 0
    jle StopJump

    ;;check out of frame
    cmp [ecx].pos.y, 0
    jle StopJump

    ;;if ok then actionOn = 2, jumpState = 1,
    mov edx, WHILE_JUMP
```

```
    mov [ecx].jumpState, edx

    ASSUME ecx: NOTHING
ret

    StopJump:
    mov ecx,adrObj
    ASSUME ecx: PTR Dmoot
    mov [ecx].actionOn, 3 ;;change actionOn
    mov ebx, [ecx].ogJumpH;;reset jumpH
    mov [ecx].jumpH, ebx
    ASSUME ecx: NOTHING
ret
shouldKeepJumping endp

gravityC proc adrObj: dword
    mov eax, adrObj
    mov ebx, 2
    cmp (Dmoot PTR [eax]).actionOn, ebx
    je JUMPING

    mov ebx,-1
    cmp (Dmoot PTR [eax]).drc.y,ebx
    je STARTJUMPING

    ASSUME eax: nothing
    invoke checkPoints , adrObj, offset ctrlPts,16, groundColor, errorColor
    cmp eax,0
    je FALLING
ret

    JUMPING:
    mov edx, PRE_JUMP
    cmp (Dmoot PTR [eax]).jumpState, edx
    jne WHILEJUMPING
    ;;not moving at all
    invoke applyJump, adrObj
ret
    WHILEJUMPING:
    ;;skip gravity
    invoke shouldKeepJumping, adrObj
    mov eax, adrObj
    cmp (Dmoot PTR [eax]).actionOn,2 ;;if not jumping then apply gravity
    jne FALLING
    ;jumping
    invoke applyJump, adrObj
ret

    STARTJUMPING:
    mov ebx,2
    mov (Dmoot PTR [eax]).actionOn, ebx
    ;;apply jump
    invoke setJumpState, adrObj, PRE_JUMP
    invoke applyJump, adrObj
    ;;skip gravity
ret
```

```
    FALLING:
    mov eax, adrObj
    mov edx, 1
    mov (Dmoot PTR [eax]).drc.y, edx
ret
gravityC endp

zukoSetAction proc adrObj: dword
    mov edi, adrObj
    mov ecx, (Dmoot PTR [edi]).actionOn ; ecx = lastAction
    mov esi, (Dmoot PTR [edi]).facing ; esi = lastFacing

    cmp ecx, 4
    je SHOOT

    mov ebx, (Dmoot PTR [edi]).drc.x
    cmp ebx,1
    je RIGHT
    cmp ebx,-1
    je LEFT
    ReturnX:

    mov edx, (Dmoot PTR [edi]).drc.y
    cmp edx, 1
    je FALLING
    cmp edx,-1
    je JUMPING
    cmp ecx, 2
    je JUMPING
    ReturnY:

    cmp ebx, 0
    je NoX
    ReturnNoX:

    cmp ecx, (Dmoot PTR [edi]).actionOn
    jne ChangedAction

    ;sameAction
    cmp esi, (Dmoot PTR [edi]).facing
    jne ChangedFacing
ret

    RIGHT:
    mov (Dmoot PTR [edi]).facing, 0
    mov (Dmoot PTR [edi]).actionOn,1
    jmp ReturnX

    LEFT:
    mov (Dmoot PTR [edi]).facing,1
    mov (Dmoot PTR [edi]).actionOn,1
    jmp ReturnX

    FALLING:
    mov (Dmoot PTR [edi]).actionOn,3
    jmp ReturnY
```

```
    JUMPING:
    mov (Dmoot PTR [edi]).actionOn, 2
    mov eax, PRE_JUMP
    cmp (Dmoot PTR [edi]).jumpState, eax
    je PreJump
    ;invoke applyJump, offset zuko1
    jmp ReturnY

    PreJump:
    ;;freezing
ret

    NoX:
    cmp edx,0
    je Standing
    jmp ReturnNoX

    Standing:
    mov (Dmoot PTR [edi]).actionOn,0
    jmp ReturnNoX

    SHOOT:
ret

    ChangedAction:
    invoke zukoResetAction, adrObj,cx;reset lastAction
ret

    ChangedFacing:
    invoke zukoResetAction,adrObj, ecx;reset lastAction
ret
zukoSetAction endp

moveFireBall proc adrObj: dword
    mov ebx, adrObj
    ASSUME ebx: PTR Dmoot
    cmp [ebx].ball.isShowing, 0
    je notShowing
    ;;does show
    cmp [ebx].ball.fireAnim.count, 0
    jle stopFireBall

    ;;move ball
    mov edx, [ebx].ball.drcX
    add [ebx].ball.pos.x,edx
    ASSUME ebx: nothing
ret

    notShowing:
ret

    stopFireBall:
    ASSUME ebx: PTR Dmoot
    mov edx, [ebx].ball.fireAnim.ogCount
    mov [ebx].ball.fireAnim.count, edx
```

```
    mov [ebx].ball.isShowing,0
    mov [ebx].ball.picIdx, 0
    ASSUME ebx: nothing
ret
moveFireBall endp

handleZukoAdvance proc adrObj: dword
    mov ebx,adrObj
    mov edx,1
    mov (Dmoot PTR [ebx]).advance, edx
ret
handleZukoAdvance endp
zukoMove proc adrObj: dword
    mov eax, adrObj
    ;;deal with freeze
    mov edx, 2
    cmp (Dmoot PTR [eax]).actionOn, edx
    je JUMPING

    RET_JUMPING:
    invoke handleZukoAdvance, adrObj
    mov eax,adrObj
    mov ebx, (Dmoot PTR [eax]).advance
    cmp ebx,0
    je dontMove

    ;;ELSE
    invoke moveFireBall, adrObj
    mov eax, adrObj
    mov ebx, (Dmoot PTR [eax]).drc.x
    add (Dmoot PTR [eax]).pos.x, ebx

    mov edx, (Dmoot PTR [eax]).drc.y
    add (Dmoot PTR [eax]).pos.y, edx
    cmp edx,-1
    je WHILEJUMPING
    RET_WHILEJUMPING:
ret

    JUMPING:
    mov edx, PRE_JUMP
    cmp (Dmoot PTR [eax]).jumpState, edx
    je PREJUMP
    jmp RET_JUMPING

    PREJUMP:
    ;freeze

ret

    WHILEJUMPING:
    ;;jumpH--
    mov edx, 1
    sub (Dmoot PTR [eax]).jumpH, ebx
jmp RET_WHILEJUMPING

```

```
    dontMove:
ret
zukoMove endp

fireBallScreenBounds proc adrObj:dword
    mov ebx, adrObj
    cmp (Dmoot PTR [ebx]).ball.isShowing, 0
    je notShowing

    ;;is showing
    cmp (Dmoot PTR [ebx]).ball.pos.x, 0
    jl TOO_LEFT

    mov edx,1200
    sub edx, (Dmoot PTR [ebx]).ball.lastW
    cmp (Dmoot PTR [ebx]).ball.pos.x, edx
    jg TOO_RIGHT

    notShowing:
ret

    TOO_LEFT:
    mov (Dmoot PTR [ebx]).ball.pos.x, 0
ret

    TOO_RIGHT:
    mov (Dmoot PTR [ebx]).ball.pos.x, edx
ret
fireBallScreenBounds endp
screenBounds proc adrObj: dword
    mov ebx, adrObj
    cmp (Dmoot PTR [ebx]).pos.x, 0
    jl TOO_LEFT

    mov edx, 1200
    sub edx, 88
    cmp (Dmoot PTR [ebx]).pos.x,edx;;minus zuko1 width
    jg TOO_RIGHT

    RET_X:

    cmp (Dmoot PTR [ebx]).pos.y, 0
    jl TOO_HIGH

    mov edx, 700
    sub edx, zukoH
    cmp (Dmoot PTR [ebx]).pos.y,edx;;minus zuko1 width
    jg TOO_LOW

    RET_Y:
    invoke fireBallScreenBounds, adrObj
ret

    TOO_LEFT:
    mov (Dmoot PTR [ebx]).pos.x, 0
jmp RET_X
```

```
    TOO_RIGHT:
    mov edx, 1200
    sub edx, 88
    mov (Dmoot PTR [ebx]).pos.x, edx
jmp RET_X


    TOO_HIGH:
    mov (Dmoot PTR [ebx]).pos.y, 0
jmp RET_Y

    TOO_LOW:
    mov edx, 700
    sub edx, zukoH
    mov (Dmoot PTR [ebx]).pos.y, edx
jmp RET_Y
screenBounds endp

mapCol proc adrObj: dword
    ;;changes drcX acording to map objects
    mov ebx, adrObj
    mov eax, (Dmoot PTR [ebx]).drc.x

    cmp eax,1
    je RIGHT
    cmp eax, -1
    je LEFT
ret

    RIGHT:
    mov ecx, (Dmoot PTR [ebx]).adrRightPts
    mov edx, (Dmoot PTR [ebx]).maxBytsRightPts
    invoke checkPoints,adrObj, ecx,edx, groundColor, errorColor
    cmp eax,1;;colors showing
    je STOP
ret

    LEFT:
    mov ecx, (Dmoot PTR [ebx]).adrLeftPts
    mov edx, (Dmoot PTR [ebx]).maxBytsLeftPts
    invoke checkPoints,adrObj, ecx,edx, groundColor, errorColor
    cmp eax,1;;colors showing
    je STOP
ret

    STOP:
    mov ebx, adrObj
    mov (Dmoot PTR [ebx]).drc.x,0
ret
mapCol endp

; ------------------------
gotHit proc adrObj:dword
    mov eax, adrObj
    mov ebx, ballDamage
```

```
    sub (Dmoot PTR [eax]).HP, ebx
    cmp (Dmoot PTR [eax]).HP,0
    jle DEATH
ret

    DEATH:
    mov (Dmoot PTR [eax]).HP,0
    mov ebx, (Dmoot PTR [eax]).lossCode
    mov win, ebx
ret
gotHit endp
hit proc adrHit:dword, adrDef:dword
    mov eax, adrHit
    mov ebx, adrDef
    cmp (Dmoot PTR [eax]).ball.isShowing, 1
    jne NoHit

    ;;check y hit
    mov ecx,(Dmoot PTR [eax]).ball.pos.y;;ecx = yB
    sub ecx, zukoH ;; ecx = yB-zukoH
    mov edx, (Dmoot PTR [ebx]).pos.y ;;edx = yZ
    cmp edx, ecx
    jle NoHit
    ;;if(yZ>=yB-zukoH) -> noHit

    mov ecx,(Dmoot PTR [eax]).ball.pos.y;;ecx = yB
    add ecx, ballH ;; ecx = yB + hB
    cmp edx, ecx
    jge NoHit
    ;;if(yZ>=yB+hB) -> noHit

    ;;HIT_Y
    mov ecx,(Dmoot PTR [eax]).ball.pos.x;;ecx = xB
    sub ecx, zukoW;; ecx = xB-Wz
    mov edx, (Dmoot PTR [ebx]).pos.x ;;edx =xZ
    cmp edx, ecx
    jle NoHit
    ;;if(xZ<=xB-Wz) 0> noHit

    mov ecx,(Dmoot PTR [eax]).ball.pos.x;;ecx = xB
    add ecx, (Dmoot PTR [eax]).ball.lastW ;; ecx = xB + wB
    cmp edx, ecx
    jge NoHit
    ;;if(xZ>=xB+wB) -> noHit

    jmp HIT
    NoHit:
ret

    HIT:
    mov (Dmoot PTR [eax]).ball.fireAnim.count, 0
    invoke playSoundAsync, offset hd.fireBall.hitSound
    invoke gotHit, adrDef
ret
hit endp

```

```
; ------------------------

handleFireBallPicIdx proc adrObj:dword
    mov ebx, adrObj
    ASSUME ebx: PTR Dmoot
    ;;count cannot be 0 here
    ;;count--
    mov edx, 1
    sub [ebx].ball.fireAnim.count, edx
    ;;if(count%cycle==0) picIdx++
    mov eax, [ebx].ball.fireAnim.count;; eax = count
    mov edi, [ebx].ball.fireAnim.cycle;;edi = cycle
    mov edx,0
    div edi; edi = eax%edi = count%cycle
    cmp edx,0
    je incPicIdx
    ASSUME ebx: NOTHING
ret

    incPicIdx:
    ASSUME ebx: PTR Dmoot
    mov edx,4
    add [ebx].ball.picIdx, edx
    ASSUME ebx: NOTHING
ret
handleFireBallPicIdx endp

drawFireBall proc adrObj:dword
    mov ebx, adrObj
    ASSUME ebx: PTR Dmoot
    cmp [ebx].ball.isShowing,0
    je notShowing

    mov ecx, [ebx].ball.picIdx
    mov edx,0
    cmp [ebx].ball.drcX, edx
    jg right
    jl left

    notShowing:
    ASSUME ebx: nothing
ret

    right:
    mov ebx,adrObj
    ASSUME ebx: PTR Dmoot
    mov eax, [ebx].ball.pos.x
    mov edx, [ebx].ball.pos.y
    invoke drd_imageDrawCrop, offset ram.fireBall.moveR, eax,edx,
      fireBallRSrcX[ecx],0,fireBallRW[ecx],ballH
    mov ecx, [ebx].ball.picIdx
    mov edx, fireBallRW[ecx]
    mov [ebx].ball.lastW, edx
    invoke handleFireBallPicIdx, adrObj
    ASSUME ebx: nothing
ret
```

```

    left:
    ASSUME ebx: PTR Dmoot
    invoke drd_imageDrawCrop, offset ram.fireBall.moveL, [ebx].ball.pos.x, [ebx].ball.pos.y, fireBallLSrcX[ecx],0,fireBallLW[ecx],zukoH
    mov ecx, [ebx].ball.picIdx
    mov edx, fireBallRW[ecx]
    mov [ebx].ball.lastW, edx
    invoke handleFireBallPicIdx, adrObj
    ASSUME ebx: nothing
ret
drawFireBall endp

displayDigit proc digit:dword, posX:dword, posY: dword

    ;; idx = digit*4
    mov eax, digit
    mov edx,0
    mov ecx, 4
    mul ecx
    ;;eax = idx
    invoke drd_imageDrawCrop, offset res.digits, posX,posY, digitSrcX[eax],0,digitW, digitH
ret
displayDigit endp

displayHP proc adrObj:dword
    mov eax, adrObj
    mov ebx, (Dmoot PTR [eax]).startPtHP.x
    mov ecx, (Dmoot PTR [eax]).startPtHP.y
    mov edx, (Dmoot PTR [eax]).HP

    loopy:
        mov edi,10
        mov eax, edx
        mov edx,0
        div edi;; edx = eax%edi = hp%10
        ;; eax = eax/edi
        push eax
        ;;digit, x, y
        invoke displayDigit, edx, ebx, ecx
        pop eax
        cmp eax,0
        je ENDLOOP
        mov edx, eax ;;edx=hp
        sub ebx, digitW
        mov eax, adrObj
        mov ecx, (Dmoot PTR [eax]).startPtHP.y
    jmp loopy
    ENDLOOP:
ret
displayHP endp

displayPlayerNum proc adrObj:dword
    mov ebx, adrObj
    mov eax, (Dmoot PTR [ebx]).playerNum
```

```
    cmp eax,1
    je PLAYER1

    PLAYER2:
    mov ecx, (Dmoot PTR [ebx]).pos.x
    add ecx, (Dmoot PTR [ebx]).toPlayerNum.x
    mov edx, (Dmoot PTR [ebx]).pos.y
    add edx, (Dmoot PTR [ebx]).toPlayerNum.y

    invoke drd_imageDraw, offset res.dig2, ecx,edx
ret

    PLAYER1:
    mov ecx, (Dmoot PTR [ebx]).pos.x
    add ecx, (Dmoot PTR [ebx]).toPlayerNum.x
    mov edx, (Dmoot PTR [ebx]).pos.y
    add edx, (Dmoot PTR [ebx]).toPlayerNum.y

    invoke drd_imageDraw, offset res.dig1, ecx,edx
ret
displayPlayerNum endp

handleZukoPicIdx proc adrObj:dword, toOgCountAdr: dword, toCountAdr: dword,
  toCycleAdr: dword
    mov ebx, adrObj
    add ebx, toCountAdr
    mov eax, [ebx]
    cmp eax,0;;if count==0 reset picIdx and count
    je resetCountAndPicIdx

    ;;count--
    mov eax, 1
    sub [ebx],eax

    ;;if(count%cycle==0) picIdx++
    mov eax, [ebx];;eax = count
    mov ebx,adrObj
    add ebx, toCycleAdr
    mov edi, [ebx];;edi = cycle
    mov edx, 0
    div edi;;edx = eax%edi = count%cycle
    cmp edx,0
    je incPicIdx
ret

    incPicIdx:
    mov ebx,4;;picIdx++
    mov eax, adrObj
    add (Dmoot PTR [eax]).picIdx, ebx
ret

    resetPicIdx:
    mov eax, adrObj
    mov (Dmoot PTR [eax]).picIdx, 0
ret

```

```
    resetCountAndPicIdx:
    mov eax, adrObj
    mov (Dmoot PTR [eax]).picIdx, 0

    ;reset count by ogCount
    ASSUME eax: nothing
    mov ebx, adrObj
    add ebx, toOgCountAdr
    mov eax, adrObj
    add eax, toCountAdr
    mov edx, [ebx]
    mov [eax], edx
ret
handleZukoPicIdx endp

zukoDrawPic proc ramImgAdr:dword, posX:dword, posY:dword, srcX:dword,
  widZZ:dword, adrObj:dword, toOgCount:dword, toCount:dword, toCycle:dword
    ASSUME eax: nothing
    invoke drd_imageDrawCrop,ramImgAdr,posX,posY,srcX,0, widZZ,zukoH
    invoke handleZukoPicIdx, adrObj, toOgCount, toCount, toCycle
ret
zukoDrawPic endp

drawZuko proc adrObj:dword
    invoke drawFireBall, adrObj
    invoke displayHP, adrObj
    invoke displayPlayerNum, adrObj

    mov eax, adrObj
    ASSUME eax:PTR Dmoot

    mov ecx, [eax].picIdx;ecx = picIdx
    mov edx, [eax].pos.x
    mov edi, [eax].pos.y

    cmp [eax].actionOn, 0
    je standState
    cmp [eax].actionOn,3
    je fallState
    cmp [eax].actionOn, 2;jumping
    je Jump
    cmp [eax].drc.x,1
    je runRState
    cmp [eax].drc.x,-1
    je runLState
    cmp [eax].actionOn, 4
    je shoot
ret

    standState:
    cmp [eax].facing, 0
    je standR
    ;;left
    invoke zukoDrawPic,offset ram.zuko.standL, edx, edi,zukoStandLSrcX
      [ecx],zukoStandWid, adrObj, Dmoot.standAnim.ogCount,
      Dmoot.standAnim.count, Dmoot.standAnim.cycle
```

```
ret

    standR:
    invoke zukoDrawPic,offset ram.zuko.standR, edx, edi,zukoStandRSrcX [ecx],zukoStandWid, adrObj, Dmoot.standAnim.ogCount, Dmoot.standAnim.count, Dmoot.standAnim.cycle
ret

    runRState:
    invoke zukoDrawPic,offset ram.zuko.runR,edx,edi,zukoRunRSrcX[ecx], zukoRunRW[ecx],adrObj,  Dmoot.runAnim.ogCount, Dmoot.runAnim.count, Dmoot.runAnim.cycle
ret
    runLState:
    invoke zukoDrawPic,offset ram.zuko.runL,edx,edi,zukoRunLSrcX[ecx], zukoRunLW[ecx],adrObj,  Dmoot.runAnim.ogCount, Dmoot.runAnim.count, Dmoot.runAnim.cycle
ret

    fallState:
    cmp [eax].facing,0
    je fallR
    ;;left
    invoke zukoDrawPic,offset ram.zuko.fallL,edx,edi,zukoFallLSrcX[ecx], zukoFallLW[ecx],adrObj, Dmoot.fallAnim.ogCount, Dmoot.fallAnim.count, Dmoot.fallAnim.cycle
ret

    fallR:
    invoke zukoDrawPic,offset ram.zuko.fallR,edx,edi,zukoFallRSrcX[ecx], zukoFallRW[ecx],adrObj, Dmoot.fallAnim.ogCount, Dmoot.fallAnim.count, Dmoot.fallAnim.cycle
ret
    Jump:
    cmp [eax].facing,0
    je jumpR
    ;;left
    ;;split pre-while
    mov esi, PRE_JUMP
    cmp [eax].jumpState, esi
    je preJumpL
    ;;jumpL
    invoke zukoDrawPic,offset ram.zuko.jumpL,edx,edi,zukoJumpLSrcX[ecx], zukoJumpLW[ecx],adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count, Dmoot.jumpAnim.cycle
ret

    preJumpL:
    invoke zukoDrawPic,offset ram.zuko.jumpL,edx,edi,zukoPreJumpLSrcX, zukoPreJumpLW,adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count, Dmoot.jumpAnim.cycle
ret
    jumpR:
    mov esi, PRE_JUMP
    cmp [eax].jumpState, esi
    je preJumpR
```

```
    invoke zukoDrawPic,offset ram.zuko.jumpR,edx,edi,zukoJumpRSrcX[ecx],
      zukoJumpRW[ecx],adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
      Dmoot.jumpAnim.cycle
ret

    preJumpR:
    invoke zukoDrawPic,offset ram.zuko.jumpR,edx,edi,zukoPreJumpRSrcX,
      zukoPreJumpRW,adrObj, Dmoot.jumpAnim.ogCount, Dmoot.jumpAnim.count,
      Dmoot.jumpAnim.cycle
ret

    shoot:
    cmp [eax].facing, 0
    je shootR
    ;;shoot L
    invoke zukoDrawPic,offset ram.zuko.shootL,edx,edi,zukoShootLSrcX[ecx],
      zukoShootLW[ecx],adrObj,  Dmoot.shootAnim.ogCount,
      Dmoot.shootAnim.count, Dmoot.shootAnim.cycle
ret

    shootR:
    invoke zukoDrawPic,offset ram.zuko.shootR,edx,edi,zukoShootRSrcX[ecx],
      zukoShootRW[ecx],adrObj,  Dmoot.shootAnim.ogCount,
      Dmoot.shootAnim.count, Dmoot.shootAnim.cycle
ret
drawZuko endp

;-----------------------------
winScreen proc adrObj1: dword, adrObj2: dword
    mov eax, win
    mov ebx, adrObj1
    cmp eax, (Dmoot PTR [ebx]).playerNum
    je PLAYER1WON

    PLAYER2WON:
    invoke drd_imageDraw,offset res.win2,0,0

ret
    PLAYER1WON:
    invoke drd_imageDraw,offset res.win1,0,0

ret
winScreen endp

resetGame proc adrObj: dword
    mov eax, adrObj
    mov ebx, (Dmoot PTR [eax]).startPos.x
    mov (Dmoot PTR [eax]).pos.x, ebx

    mov ebx, (Dmoot PTR [eax]).startPos.y
    mov (Dmoot PTR [eax]).pos.y, ebx

    mov ebx, StartHP
    mov (Dmoot PTR [eax]).HP, ebx

    mov ebx, (Dmoot PTR [eax]).actionOn
```

```
    invoke zukoResetAction, adrObj, ebx
ret
resetGame endp
```

```
include \masm32\include\user32.inc;;sound
includelib \masm32\lib\user32.lib

include \masm32\include\winmm.inc;;sound
includelib \masm32\lib\winmm.lib

.code
zukoDflt proc adrObj:dword
    mov ebx, adrObj
    ASSUME ebx: PTR Dmoot
    mov [ebx].drc.x, 0
    mov [ebx].drc.y,0
    ASSUME ebx: nothing
ret
zukoDflt endp

zukoActByKey proc adrObj: dword ,toKeyAdr: dword, toDrcAdr:dword, valuu:dword
    ;;if pressed insert value
    mov ebx, adrObj
    add ebx, toKeyAdr
    invoke GetAsyncKeyState, [ebx]
    cmp eax,0
    jne PRESSED;jne
ret

    PRESSED:
    mov edx, valuu
    mov edi, adrObj
    add edi, toDrcAdr
    mov [edi], edx
ret
zukoActByKey endp

checkPoints proc adrObj:dword, adrArr:dword,sizeofArr:dword, color1:dword,
  color2: dword
;                                offset of arr[0], max bytes in arr , bgr, bgr

;result in eax -> 0-colors don't show, 1- one or both colors show

    invoke drd_beginHdc;get handle
    mov esi, eax ; esi = handle
    mov ecx,0;ecx = idx
    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
        jl dloop;if(ecx+8<sizeofArr) stay in loop

    NOTSHOW:
    mov eax,0
    invoke drd_endHdc, esi
ret

    SHOW:
    invoke drd_endHdc, esi
    mov eax,1
ret
checkPoints endp

playSoundAsync proc adrPath:dword
    invoke PlaySound , adrPath,NULL, SND_ASYNC
ret
playSoundAsync endp
```