פרויקט באסמבלי

שם : מנואל נמירובסקי

כיתה : י'9

ת.ז : 211338108

מורה : יהודה אור

שפה וסביבת עבודה : אסמבלי x86 , Visual studio

שם המשחק : Robin Hood For The Neighborhood

תוכן עניינים

הקדמה

אסמבלי היא שפת התכנות הכי בסיסית שקיימת (שעדיין מובנת לבני אדם), והכי קרובה לשפת מכונה, שהיא השפה שהמחשב קורא ומבין לבסוף כשהוא מריץ תוכניות. באסמבלי ההוראות הם מאוד בסיסיות ומתורגמות באופן ישיר להוראות בשפת מכונה. בשפה עילית רגילה כל הוראה מתקמפלת לאוסף די גדול של הרבה הוראות קטנות, אבל באסמבלי אתה ממש כותב את ההוראות האלו באופן ידני. לכן תכנות בשפה הזאת נחשב קשה יותר מתכנות בשפה רגילה: הקוד הרבה פחות אינטואיטיבי וכדי לעשות פעולות יחסית בסיסיות צריך כמה הוראות. (לדוגמא, כדי להדפיס "Hello World" בשפה עילית, אנו זקוקים לשורה אחת, ובאסמבלי – 7 שורות!).

בנוסף, בגלל הפרטים שאפשר להגיע אליהם בשפה הזאת, התכנית שנכתבת יכולה להיות מאוד יעילה ומהירה: אתה שולט בכל מה שקורה במחשב ואתה יודע בדיוק מה קורה בכל רגע, לכן היום השימוש באסמבלי הוא בעיקר במערכות הפעלה, או בתוכניות שצריכות להיות מהירות במיוחד.

תהליך הלמידה

הלימוד של השפה הזאת הוא מעניין במיוחד. בניגוד לשפה עילית רגילה, אתה לא לומד אוסף של כללי תחביר. אתה צריך להבין איך המחשב עובד וזה גם גורם לך לראות שהרבה דברים הם יותר מסובכים ופחות טריוויאליים ממה שחשבת. (לדוגמא , קריאה וחזרה מפונקציות).

שפת אסמבלי הינה שפה שונה משאר השפות שכתובות כיום, היא שפה גדולה יותר וכבדה יותר מבחינת לימוד, מכיוון שכל החומר הנלמד הוא יותר מבחינה תאורטית מאשר תחביר. בשפת אסמבלי אתה צריך להבין איך הכל עובד ומה קורה מעבר לתחביר הרגיל שאתה מקבל בשפה עילית.

הסבר על הוראות ואוגרים

אוגרים

האוגרים הם יחידות זכרון בני 16 ביט כל אחת היושבות במעבד ומבצעות את כל עבודות החישוב. יחידת האוגרים מהירה הרבה יותר מהזיכרון הרגיל של המחשב, דבר שהופך את פעולת המעבד למהירה יותר.

בנוסף לאוגרים הרגילים שיש ב32 ביט (EAX , EBX, ECX, EDX), יש גם אוגרים אחרים (כמו ESP או EIP) שיש להם תפקיד מיוחד, כמו למשל לשמור את הכתובות של סוף המחסנית, ואסור לעשות בהם שימוש נוסף.

הוראות

באסמבלי מספר ההוראות האפשריות הוא יחסית קטן. מותר לעשות רק פעולות בסיסיות כמו לאחסן מידע באוגרים, לבצע פעולות חשבוניות עם האוגרים, להעביר מידע מהזיכרון (RAM), או לקפוץ להוראה במקום אחר. הנה דוגמא:

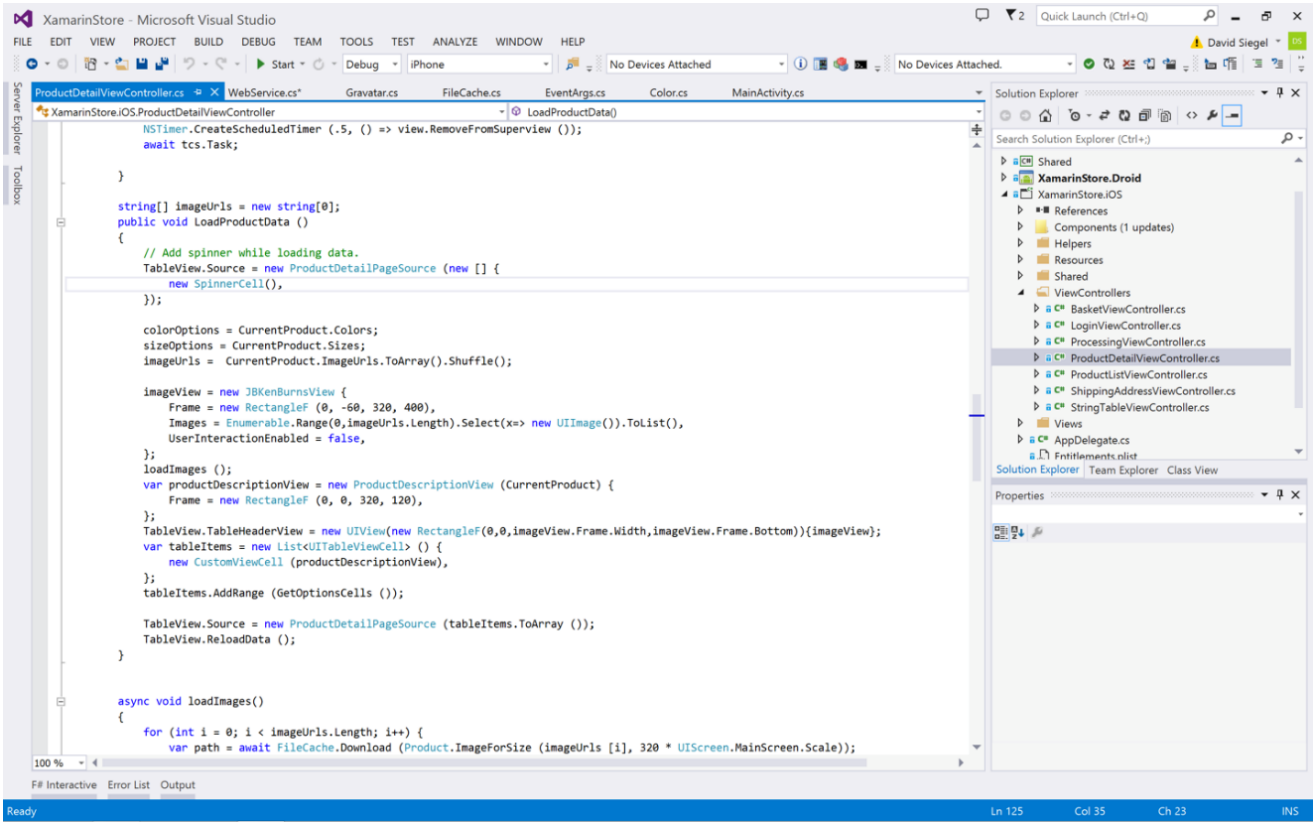
```
cmp eax , ebx
```

פקודה זו משווה בין המספר שמאוחסן באוגר EAX, לבין המספר שמאוחסן באוגר EBX. בעזרת הפקודה הזו אני יכול לעשות כל מיני דברים שאני רוצה בשביל השוואות. כמו לדוגמא לזמן לולאה ברגע שהמספרים האלו שווים או לזמן לולאה ברגע שאני רואה שמהספרים האלו לא שווים.

סביבת העבודה

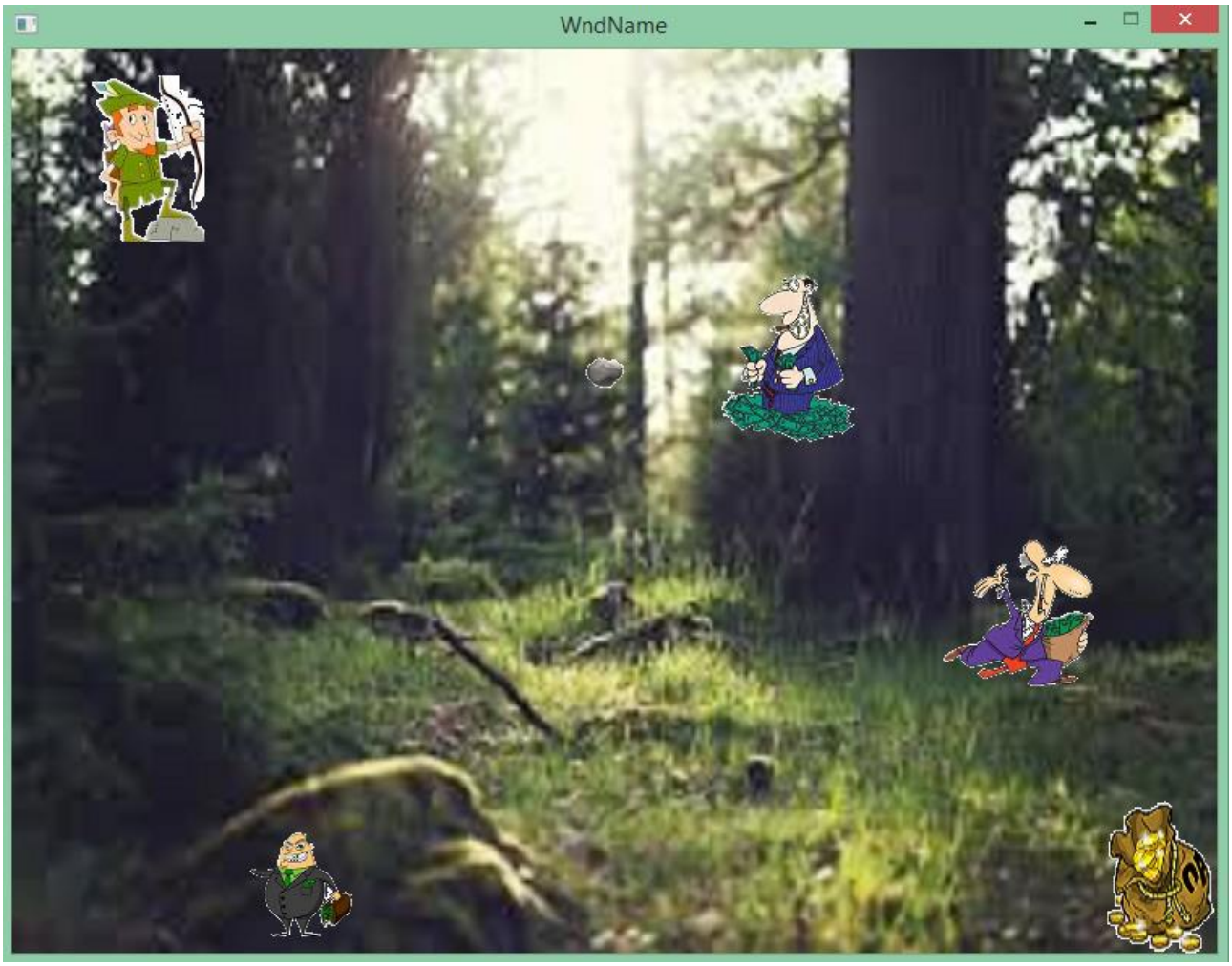
Visual Studio היא סביבת פיתוח מובילה מבית מיקרוסופט. המאפשרת למתכנתים לפתח תוכניות מחשב ואתרי אינטרנט. הגרסא הרשמית האחרונה היא ויז'ואל סטודיו 2015, המיועדת לפיתוח יישומים עבור סביבת win32 ואפליקציות של windows, וגם עבור NET Framework.

נורא שמחתי שיש לנו אפשרות לעבוד בסביבה הזו מכיוון שאני מכיר את סביבת הפיתוח הזו ויצא לי לעבור בה כבר בפרויקטים אחרים. סביבת העבודה נורא נוחה כאשר מפעילים דיבאגר (צורת דיבוג שבנויה על העיקרון של step by step, שלב אחר שלב בקוד), קל נורא לראות את מבני התיקיות, נוח נורא לעשות ספריות, נוח לעבור על כל מיני פרויקטים שעשית והקוד נורא קריא.



המשחק שלי

כאשר סיפרו לנו על הפרויקט הסופי שלנו שאתו אנו ניגשים לבגרות, רציתי לעשות משחק שמשלב את כל הנושאים שלמדנו. חיפשתי משחק שבו אני אוכל לשלב כל מיני נושאים, כמו התנגשויות בין שחקנים, שילוב תמונות ויצירת user interface נוח וקריא. לאחר מחשבה ארוכה החלטתי לבסוף ליצור משחק שבו יש את המשתמש, שמשחק ברובין הוד שנמצא בתוך היער וצריך להגיע לשק זהב שמונח בצד השני של המפה. מטרתו של רובין הוד היא לקחת את השק כדי שיוכל לחלק כסף לכל האנשים העניים שנמצאים בכפר ליד היער, אך על אותו שק זהב שרובין הוד צריך לקחת, שומרים 3 אנשים עשירים שגנבו את אותו שק זהב ושמו אותו ביער. אותם העשירים מסתובבים במסלול מסוים וקבוע כדי שיוכלו לשמור על אותו שק זהב. מטרתו של רובין הוד היא לקחת את שק הזהב בכל מחיר, גם אם זה אומר לירות על אותם אנשים עשירים...



להציג תמונה במסך

אחד הדברים הראשונים שעשיתי בתחילת הפרויקט היה לאסוף תמונות בשביל המשחק ולחפש פקודות שאיתם אני אוכל לעבוד כדי לגרום לתמונות להופיע על המסך. לכן, כאשר קיבלנו את התיקייה drd.inc, חיפשתי שם פקודות שאיתם אני אוכל להציג תמונה בצורה שאני רוצה שהיא תיהיה על המסך. קודם כל, לפני שיהיה אפשר להשתמש בפקודות, היה עליי להכניס לחלק הdata משתנים שבתוכם אכיל את כל המידע שאני צריך לכל תמונה

```
aGold BYTE "gold.bmp",0
gold Img<0,0,0,0>
```

שתי השורות האלו מחזיקות לי את הנתונים שאני צריך בשביל להראות תמונה על המסך. המשתנה הראשון מראה את השם של הקובץ שבתוכו נמצאת התמונה, והמשתנה השני מחזיק מבנה של תמונה עם ערכים שצריך לאתחל

לאחר מכן, עברתי על תיקיית drd.inc שקיבלנו בתחילת הפרויקט, וכאשר חיפשתי שם פקודות, מצאתי את הפקודות הבאות שיהיו שימושיות ביצירת תמונה במסך :

```
; load an image from file `filename` into an Img struct
drd_imageLoadFile PROTO filename:DWORD, pimg:DWORD
```

פקודה זו לוקחת את השם של הקובץ, ואת המבנה שהגדרנו בשביל התמונה, ומטעינה את התמונה בתוך הפרויקט כדי שאוכל לעשות שימוש בה.

```
; draw the image on the back buffer, starting at coord
;(dstX,dstY). must be outside drd_pixelsBegin
drd_imageDraw PROTO pimg:DWORD, dstX:DWORD, dstY:DWORD
```

פקודה זו מציגה את התמונה במסך לפי קוארדינציות x,y שאנחנו מחליטים.

```
; set the color which will be interpreted as transparency
drd_imageSetTransparent PROTO pimg:DWORD, color:DWORD
```

פקודה זו מורידה את הרקע של הצבע שאנו מכניסים לה, השתמשתי בפקודה זו מכיוון שהתמונות שהכנסתי למשחק היו תמונות עם רקע לבן, כך שבעזרת הפונקציה, הייתה לי האפשרות להוריד את הרקע ולהבליט את הדמות עצמה.

להזיז את רובין הוד

פונקציה נורא משמעותית בפרויקט הייתה הפונקציות שגורמת לתזוזה של רובין הוד. תפקידו של רובין הוא להגן על עצמו בזמן ההתקדמות שלו לעבר שק הזהב ולאסוף את השק. כדי לעשות את התפקיד שלו הוא יכול לירות חיצים על הדמויות כדי לפנות לו את דרכו לכיוון שק הזהב.

כדי לגרום לרובין לזוז ולירות, השתמשתי במקלדת. כדי להשתמש במקלדת צריך להשתמש בGetAsyncKeyState, שאליו מכניסים את התו במקלדת שאנו רוצים לראות מה מצבו, וכאשר הוא נהפך ל1, אנו יודעים שהוא משומש וכך אפשר לקרוא ללולאה שתגרום לרובין לזוז עד לגבול של המפה, או לגרום לרובין לירות.

```
moveRobin PROC
X   invoke GetAsyncKeyState, VK_RIGHT \ cmp eax, 0 \ jne right
X   invoke GetAsyncKeyState, VK_LEFT \ cmp eax, 0 \ jne left
X   invoke GetAsyncKeyState, VK_UP \ cmp eax, 0 \ jne up
X   invoke GetAsyncKeyState, VK_DOWN \ cmp eax, 0 \ jne down
X   invoke GetAsyncKeyState, VK_SPACE \ cmp eax, 0 \ jne shoot
        jmp nothing
    right:
        cmp robin_x,725
        je nothing
        inc robin_x
        ret
    left:
        cmp robin_x,0
        je nothing
        dec robin_x
        ret
    up:
        cmp robin_y,0
        je nothing
        dec robin_y
        ret
    down:
        cmp robin_y,490
        je nothing
        inc robin_y
        ret
    shoot:
        mov is_arrow, 1
        ret
    nothing:
        ret
moveRobin ENDP
```

בכל אחד מהלולאות, מזיזים את ה-X או את ה-Y (תלוי לפי הכיוון) עד לגבול של התמונה עם המפה. גודל המפה הוא 800x600, כך שכדי להזיז את רובין עד לגבול, צריך להוריד מגודל המסך את גודל התמונה של רובין כדי שהתמונה תיכנס למסך ולא תצא מחוצה לו.

Is_arrow מייצג Boolean שכאשר הוא נהפך ל1, פונקציה אחרת (שעליה אפרט בהמשך), גורמת לירייה של החץ.

כדי לגרום לרובין לירות, רשמתי פונקציה בשביל היריה. בפונקציה זו החץ מצויר כאשר ה-X שלו מתקדם כל פעם ב1, וזאת על מנת להזיז אותו לצד ימין (רובין יכול לירות רק ימינה). במהלך הפונקציה ישנם הרבה השוואות עם הפגיעה של החץ בכל דמות, וכך כאשר החץ פוגע בדמות מסוימת, גם הדמות וגם החץ צריכים להיעלם.

<u>קוד היריה</u>

```
Shoot PROC
    cmp is_arrow, 0
    je destroy_arrow

    cmp is_fired, 1
    je draw

    mov eax, robin_x
    mov ebx, robin_y

    mov is_fired, 1

    mov arrow_x, eax
    mov arrow_y, ebx

    add arrow_y , 55
    add arrow_x , 75
    draw:
        cmp arrow_x, 737
        jge destroy_arrow
        mov ebx ,arrow_x
        add ebx , 62
        invoke monster1Arrow
        invoke monster2Arrow
        invoke monster3Arrow
        inc arrow_x
        invoke drd_imageDraw, offset arrow, arrow_x, arrow_y
        ret
        destroy_arrow:
        mov is_arrow, 0
        mov is_fired, 0
        ret
Shoot ENDP
```

שתי המשתנים האלו מייצגים לי Boolean-ים. אחד מהם בודק אם אני יכול לירות והשני בודק אם אני כבר יריתי את החץ.

בהתחלה, כדי לגרום לחץ להופיע היישר מתמונתי של רובין הוד, אנו צריכים להזיז את המשתנים של Y,X של החץ ולמקם אותם במיקום שאנחנו רוצים (שזה באמצע הגוף של רובין הוד). בשביל זה הכנסתי לתוך האוגרים eax,ebx את המיקום של רובין הוד ולאחר מכן הכנסתי את המידע בתוך האוגרים למשתני הX ,Y של החץ ולאחר מכן להזיז את החץ לפי גודל הדמות של רובין הוד כדי שנוכל לירות את החץ מאמצע גופו של רובין הוד.

בעמוד הבא ישנה דוגמא לאחד מהפונקציות האלו.

כדי להרוס את החץ צריך להפוך את שתי המשתנים הבוליאניים ל-0 ואז הפונקציה לא תיכנס ללולאה שמציירת את החץ

דוגמא של בדיקה של פגיעת החץ באחד מהעשירים

הכנסתי לתוך האוגרים eax,ecx את קוארדינציות ה-X וה-Y של החץ ולאחר מכן אני מכניס לתוך האוגר ebx את ה-X של העשיר השני כדי לבדוק השוואה בערכי ה-X ולאחר מכן, כאשר אני נכנס לתוך הלולאה שבודקת את ה-Y אני משנה את ערכו של האוגר ebx לערך ה-Y של העשיר השני (כדי לחסוך בעוד אוגרים). השימוש באוגרים נובע מכך שאין אפשרות להשוות בין שתי משתנים ולכך צריך שימוש באוגרים.

לאחר שאני משווה את ערכי ה-X של העשיר ושל החץ, אני בודק את ערך הY העליון ואת ערך הY התחתון בשביל לקבל רק את הטווח של המפלצת, כי אם נשווה רק טווח אחד מה שירה הוא שכאשר נגיע לטווח הזה (בין אם פגענו בעשיר או לא), העשיר ייעלם למרות שלא פגענו בו.

לאחר שיצרתי פגיעה מדויקת בדמות של העשיר, הגיע הזמן להעלים אותו מהמסך ולהעלים את אותו החץ שבו יריתי. בשביל זה יצרתי שתי לולאות. הלולאה הראשונה היא בשביל לשנות Boolean שאומר אם הדמות מתה או לא, והלולאה השנייה היא לשנות את שתי הboolean-ים שאיתם אוכל להעלים את החץ.

```
monster1Arrow PROC

    mov ecx , arrow_y
    mov eax , arrow_x
    add eax , 43

    mov ebx , mon1_x
    cmp ebx , eax
    je mon1checkY

    jmp nothing
    mon1checkY:
        mov ebx , mon1_y
        cmp ecx , ebx
        jge mon1checkSecY
        ret
    mon1checkSecY:
        add ebx , 69 ; for comparing the y of monster
        cmp ecx , ebx
        jbe destroyMon1
        ret
    destroyMon1:
        mov firstRichAlive , 0
        jmp destroy_arrow
        ret
    destroy_arrow:
        mov is_arrow, 0
        mov is_fired, 0
        ret
    nothing:
        ret

monster1Arrow ENDP
```

איך אפשר להיפסל

כדי לגרום למשחק להיות יותר מהנה, הייתי צריך לחשוב על דרכים שבהם רובין הוד יכול למות ובכך לגרום להפסד. כמובן שהאפשרות שרובין הוד ייגע באחד מהעשירים שבדרכו לשק הזהב, יוביל להפסד, אך רציתי להוסיף עוד אפשרות למשחק שבה רובין הוד יפסל והמשתמש יוכל להפסיד.

דוגמא לאחד מהקודים להשוואת הערכים של רובין הוד ושל אחד מהעשירים שיוביל לפסילה

```
hitMonster2 PROC

    mov ecx , robin_y
    mov eax , robin_x

    mov ebx , mon2_x
    cmp ebx , eax
    je mon2CheckSecx

    mon2CheckSecx:
        add eax , 75
        cmp ebx , eax
        jbe mon2CheckFirstY
        ret
    mon2CheckFirstY:
        mov ebx , mon2_y
        cmp ecx , ebx
        jbe mon2CheckSecY
        ret
    mon2CheckSecY:
        add ecx , 110 ; for comparing the y of robins
        cmp ecx , ebx
        jge youLose
        ret
    youLose:
        mov is_robin_lose , 1
        ret

    nothing:
        ret

hitMonster2 ENDP
```

בכל אחד מהפונקציות, האוגרים eax,ecx מקבלים את המיקום של רובין הוד לפי ציר ה-X ולפי ציר ה-Y. בנוסף לכך, אני משתמש גם באוגר הebx במטרה שיכיל לי את המיקומים של הדמות אותה אני רוצה לבדוק.

לאחר שערכתי השוואה בין ערכי ה-X של רובין הוד והדמות העשירה, אני מבצע השוואה שניה של ערכי ה-X רק שהפעם אני מוסיף לערך ה-X של רובין 75 (שזה הגודל של רובין בציר ה-X). כל זה במטרה לבדוק האם הדמות נמצאת בטווח ה-X-ים של רובין הוד. (אם הדמות נמצאת בטווח אני יוצר פקודת jump below).

את אותה השוואה שערכתי בטווחי ה-X, אני יוצר גם בין טווחי ה-Y על מנת לוודא שהדמות העשירה ורובין הוד נמצאים ממש אחת על השנייה ורק אז יהיה אפשר עבור ללולאה שמכריזה על הפסד המשתמש.

בתוך אותה לולאה אני משנה את ערכו של Boolean שמייצג לי האם רובין הפסיד. אם הלולאה נהפכת ל1, אני יוצר קפיצה בmain ללולאה אחרת שמשנה לי את המוזיקה למוזיקה של הפסד ואת רקע המשחק לרקע של הפסד.

כמו שעשיתי עם רובין הוד, שיכול לירות חיצים על הדמויות העשירות כדי להגיע לשק הזהב, נתתי אפשרות לאחד העשירים לנסות להרוג את רובין הוד בעזרת זריקת אבנים על הדמות. ברגע שאחת מהאבנים שהדמות זורקת על רובין הוד יפגעו בו, רובין הוד ימות והמשתמש יפסיד.

קוד לזריקת האבנים מאחת הדמויות של האנשים העשירים

```
monster2Shooting PROC
    cmp is_mon2_fired, 1
    je drawRock

    cmp boolean_is_rock_no_need , 1
    je drawThisRock

    mov eax, mon2_x
    mov ebx, mon2_y
    mov is_mon2_fired, 1

    mov rock_x, eax
    mov rock_y, ebx
    add rock_y , 55
    sub rock_x , 88
    jmp nothing
    drawRock:
        cmp rock_x, 0
        je destroy_Rock

        invoke checkRockAroow

        dec rock_x
        invoke drd_imageDraw, offset rock, rock_x, rock_y
        ret
    drawThisRock:
        ret
    destroy_Rock:
        mov is_mon2_fired, 0
        ret
    nothing:
        ret
monster2Shooting ENDP
```

משתנה בוליאני שיבדוק אם הדמות השנייה ירתה אבן ויעביר את הקוד ללולאה של ציור האבן

משתנה בוליאני שיבדוק האם יש צורך בלצייר אבנים (מכיוון שאם הדמות השנייה מתה, אין סיבה להמשיך לצייר את האבנים).

כמו שעשינו בקוד של יריית החיצים מרובין הוד, אנו מכניסים לתוך האוגרים eax,ebx את המיקום של הדמות העשירה השניה (שממנה אנו נראה את האבנים) ואז מכניסים את הנתונים האלו לתוך המשתנים של האבנים ומזיזים את המשתנים של האבנים לפי גודל הדמות (כדי ליצור ירייה מאמצע הגוף).

בתוך לולאה הציור של האבן אני מקטין את ערך ה-X של האבן (מכיוון שהאבן זזה לכיוון שמאלה) ומצייר את המיקום החדש שלה עד שאני מגיע ל=0x ואני עובר ללולאה שמוחקת לי את האבן

מחיקת האבן נעשית על ידי המשתנה הבוליאני שאומר לנו האם להיכנס לתוך הלולאה של ציור האבן. אנו משנים את ערכו ובכך גורמים להתחלה חדשה של האבן בדיוק המיקום שבו הדמות זזה

לאחר שיצרתי פונקציה לזריקת האבנים לכיוון רובין הוד, הייתי צריך ליצור פונקציה שתבדוק מתי האבן פוגעת ברובין הוד

פונקציית הבדיקה לפגיעה ברובין עם האבן

```
hitRock PROC

    mov ecx , robin_y
    mov eax , robin_x

    mov ebx , rock_x
    cmp ebx , eax
    jge checkRockSecx

    jmp nothing
    checkRockSecx:
        add eax , 75
        cmp ebx , eax
        jbe checkRockFirstY
        ret
    checkRockFirstY:
        mov ebx , rock_y
        cmp ecx , ebx
        jbe checkRockSecY
        ret
    checkRockSecY:
        add ecx , 110 ; for comparing the y of robins
        cmp ecx , ebx
        jge youLose
        ret
    youLose:
        mov is_robin_lose , 1
        ret
    nothing:
        ret

hitRock ENDP
```

כמו ההשוואה בין המיקום של אחד הדמויות העשירות לבין המיקום של רובין הוד, כך גם נעשית ההשוואה בין המיקום של האבן לבין המיקום של רובין הוד. אני מכניס לתוך האוגרים eax,ecx את קואורדינציות X,Y של רובין הוד במטרה להשוות אח"כ עם הערכים של האבו.

לאחר שערכתי השוואה בין ערכי ה-X של רובין הוד והאבן, אני מבצע השוואה שניה של ערכי ה-X רק שהפעם אני מוסיף לערך ה-X של רובין 75 (שזה הגודל של רובין בציר ה-X). כל זה במטרה לבדוק האם האבן נמצאת בטווח ה-X-ים של רובין הוד. (אם האבן נמצאת בטווח, אני יוצר פקודת jump below).

את אותה השוואה שערכתי בטווחי ה-X, אני יוצר גם בין טווחי ה-Y על מנת לוודא שהאבן ורובין הוד נמצאים ממש אחד על השני, ורק אז יהיה אפשר עבור ללולאה שמכריזה על הפסד המשתמש.

בתוך אותה לולאה אני משנה את ערכו של Boolean שמייצג לי האם רובין הפסיד. אם הלולאה נהפכת ל1, אני יוצר קפיצה בmain ללולאה אחרת שמשנה לי את המוזיקה למוזיקה של הפסד ואת רקע המשחק לרקע של הפסד.

הזזת הדמויות העשירות

לדמויות העשירות החלטתי ליצור תזוזה קבועה בשני צורות שונות. לדמות העשירה השנייה והשלישית יצרתי תזוזה באותה הצורה ולדמות הראשונה יצרתי תזוזה ייחודית משלה.

תזוזת הדמות העשירה הראשונה

```
moveMonster1 PROC
    cmp mon1Way , 1
    jne secMov1
    cmp mon1_y , 0
    je nothing1
    inc mon1_x
    dec mon1_y
    ret
    secMov1:
        cmp mon1Way , 2
        jne thrMon1
        cmp mon1_x , 720
        jae nothing1
        inc mon1_x
        inc mon1_y
        ret
    thrMon1:
        cmp mon1Way , 3
        jne final1
        cmp mon1_y , 520
        jae nothing1
        dec mon1_x
        inc mon1_y
        ret
    final1:
        cmp mon1_x , 0
        je endd1
        dec mon1_x
        cmp mon1_x , 0
        ret
    nothing1:
        inc mon1Way
        ret
    endd1:
        mov mon1Way , 1
        ret
moveMonster1 ENDP
```

את הדמות העשירה הראשונה הזזתי בפגיעה בכל אחד מהפינות של המפה. הדמות מתחילה בפינה מסוימת וברגע שהיא נוגעת באחת מהפינות, היא עושה השוואה עם ציר ה-X הוא ציר ה-Y של המפה (תלוי באיזה צד היא פוגעת), ואז היא זזה לצד האחר.

בלולאת הfinal מתבצעת חזרה למיקום ההתחלתי של הדמות וקפיצה ללולאת endd1 שמחזירה את הערך של התזוזה של הדמות ל1 בכדי להתחיל תזוזה חדשה מהmove הראשון של הפונקציה.

את לולאת nothing1 יצרתי בשביל לשנות את הערך של התזוזה של הדמות, ובכדי לעשות ret לפונקציה כאשר היא לא הגיעה לגבול.

דוגמא לאחת התזוזות של שני הדמויות העשירות האחרות

```
moveMonster3 PROC
    mov eax , mon3Bool1
    mov ebx , mon3Bool2
    cmp eax , ebx
    je switch1
    jne switch3
    switch1:
        cmp mon3_x , 500
        je switch2
        dec mon3_x
        ret
    switch2:
        cmp mon3_y , 502
        je incMon3Bool1
        inc mon3_y
        ret
    incMon3Bool1:
        inc mon3Bool1
        ret
    switch3:
        cmp mon3_y , 325
        je switch4
        dec mon3_y
        ret
    switch4:
        cmp mon3_x , 700
        je incMon3Bool2
        inc mon3_x
        ret
    incMon3Bool2:
        inc mon3Bool2
        ret
moveMonster3 ENDP
```

תזוזה זו עובדת בצורה של תנועה ל-2 כיוונים, ואז אותה התנועה בצורה הפוכה (לצד שמאל ולמטה ולאחר מכן למעלה ולצד ימין). כדי ליצור תנועה זו נעזרתי בשתי משתנים בוליאנים שכאשר הם יהיו שווים, אני עובר ללולאה של תחילת התנועה וכאשר הם לא שווים אני עובר ללולאה של תחילת התנועה לצד השני.

שתי הלולאות הראשונות מזיזות את הדמות לכיוון שמאלה ולמטה, והתנאי שמעבר מלולאה אחת לשנייה הוא הגבול השמאלי שהדמות יכול לזוז אליו. בלולאה השנייה התנאי עצירה הוא כלפי הגבול התחתון, ולולאה זו קוראת ללולאה שמגדילה את אחד מהמשתנים הבוליאניים ובכך גורמת להם להיות שונים.

שתי התנועות הבאות עובדות על אותו עיקרון רק שבסוף התנועה הרביעית, אני מעלה את המשתנה הבוליאני השני ב-1 ובכך יוצר שוויון בין שתי המשתנים. שוויון זה מוביל לקפיצה ללולאה הראשונה שתתחיל את התנועה שוב.

מחיקת הדמויות העשירות מהמסך

לאחר שהמשתמש פוגע בירייה על אחת הדמויות, הדמות צריכה להיעלם מהמסך. אם הדמות לא תיעלם ואני פשוט אדלג על החלק שבו אני מצייר אותה בלולאת הmain שלי, אז המיקום שלה עדיין יישמר ואז כשאני אעבור על בדיקת המיקום של רובין ושל אותה דמות, רובין יוכל להיפגע ממנה ולהפסיד למרות שהוא ירה עליה. כדי למחוק את הדמות מחיקה טוטאלית ולדאוג לכך שרובין הוד לא ייפגע מהדמות, השתמשתי בפונקציית drd_imageDelete שמוחקת את התמונה ואי אפשר להשתמש בתמונה הזו שוב אלא אם כן נשתמש שוב בפונקציית drd_imageLoadFile אך מכיוון שלא היה לי שימוש חוזר בדמות שרובין ירה בה, לא הטענתי את הדמות שוב פעם.

<u>דוגמא למחיקת דמות עשירה מהמשחק</u>

```
checkIfMon3Dead PROC

    cmp thirRichAlive, 1
    jne movThrRich

    ret

    movThrRich:

        cmp imgDelMon3 , 1
        je noDelAgain3

        invoke drd_imageDelete, offset monster3
        mov imgDelMon3 , 1

        noDelAgain3:

        mov mon3_x , 700
        mov mon3_y , 0
        ret
checkIfMon3Dead ENDP
```

אני מבצע השוואה של משתנה בוליאני שאומר לי אם הדמות מתה או לא. אם המשתנה לא שווה ל-1 זה אומר שהדמות מתה וצריך לעבור ללולאה שמוחקת אותה

בתוך הלולאה הזו ישנו עוד משתנה בוליאני שבודק האם מחקנו את התמונה. מחיקת התמונה אמורה להתבצע פעם אחת, כך שאם נמחק שוב תמונה שכבר מחקנו, המשחק יקרוס.

לאחר שמחקתי את התמונה, אני מזיז את הערכים של אותה התמונה לצד של המפה כדי שלא תהיה התנגשות עם הערכים.

ישנה דמות אחת שבה אני מבצע מחיקה שונה. בגלל שהדמות העשירה השנייה יורה אבנים, כאשר רובין הוד הורג אותה, צריך שהאבנים יפסיקו לירות.

קוד מחיקת יריית האבנים

```
checkIfRockDead PROC

    cmp boolean_is_rock_no_need , 1
    je deleteRock

    ret

    deleteRock:
        cmp imgDelRock1 , 1
        je noDelAgainRock

        invoke drd_imageDelete, offset rock

        mov imgDelRock1 , 1

        noDelAgainRock:

        mov  is_mon2_fired , 0
        mov rock_x , 775
        mov rock_y , 0

        ret
checkIfRockDead ENDP
```

תחילה, אני משתמש במשתנה בוליאני שבודק אם יש לי צורך בציור האבנים (הרגע היחיד שבו לא יהיה לי צורך באבנים זה אם הדמות העשירה השנייה מתה). כאשר המשתנה הבוליאני שווה ל-1, אני עובר ללולאה של מחיקת האבן לתמיד.

בלולאה זו, אני עושה את אותה הבדיקה של מחיקת התמונה של האבן כמו עם הדמויות העשירות, רק שלאחר שמחקתי את האבן, אני מזיז את ערכי ה-X וערכי ה-Y שלה לפינת המסך ומשנה את המשתנה הבוליאני שאומר לי אם הדמות השנייה ירתה ל-0 ובכך יוצר דילוג מעל הלולאה שיורה ירייה.

תמונות מהמשחק

תמונת הניצחון

כדי לנצח במשחק צריך להגיע לשק הזהב, לא משנה אם הרגת את כל הדמויות או לא. כדי לבדוק האם רובין הוד הגיע לשק הזהב וניצח הכנתי פונקציה נפרדת בשביל הבדיקה.

קוד בדיקת הניצחון

```
checkIfWin PROC

    cmp robin_x , 725
    je checkFirstY

    jmp nothing

    checkFirstY:
        cmp robin_y , 415
        jge checkSecY
        ret

    checkSecY:
        cmp robin_y , 490
        jle jIfWin
        ret

    jIfWin:
        mov is_robin_win , 1
        ret

    nothing:
        ret

checkIfWin ENDP
```

אני משווה את המיקום של רובין הוד לפי ציר ה- X, אם רובין הוד הגיע לטווח ה-X שבו שק הזהב נמצא, נשאר רק לבדוק האם רובין הוד נמצא בין טווחי ה-Y של שק הזהב

לאחר שבדקתי אם רובין הוד נמצא בין טווחי ה-Y של שק הזהב, אני משתמש במשתנה בוליאני שכאשר הוא נהפך להיות 1, אני קופץ ללולאה שמשנה את המוזיקה למוזיקה אחרת ואז אני קופץ לעוד לולאה שמשנה את התמונה של המסך לתמונה של ניצחון

```
winLoop:
    invoke drd_imageDraw , offset win , 0 , 0
    invoke drd_processMessages
    invoke drd_flip
    jmp winLoop
```

הלולאה הזו מציגה את התמונה של ניצחון המשחק ויוצרת קפיצה אליה כדי שלא נתחיל שוב את לולאת הmain.

תמונת ההפסד

כאשר מפסידים במשחק, המשתנה הבוליאני is_robin_lose נהפך ל-1, כמו שהראיתי במהלך הבדיקות אם רובין נגע בדמויות.

```
loseLoop:
    invoke drd_imageDraw , offset lose , 0 , 0
    invoke drd_processMessages
    invoke drd_flip
    jmp loseLoop
```

הלולאה הזו מציגה את התמונה של הפסד המשחק ויוצרת קפיצה אליה כדי שלא נתחיל שוב את לולאת הmain.

חוקי המשחק

בתחילת המשחק, כאשר אנחנו נמצאים בדף הפתיחה, יש אפשרות לעבור לדף ההוראות והחוקים של המשחק, שנמצא בלולאה משלו בתוך הקוד.

```
rulesLoop:
    invoke drd_imageDraw , offset rules , 0 , 0
    invoke drd_processMessages
    invoke drd_flip
    X   invoke GetAsyncKeyState, 52h \ cmp eax, 0 \ jne again
    X   invoke GetAsyncKeyState, 54h \ cmp eax, 0 \ jne openGame
    jmp rulesLoop
```

בלולאת החוקים אנו משתמשים במקלדת כדי לדעת מתי אנחנו צריכים לצאת מהדף ואו לחזור לדף פתיחת המשחק, או פשוט להתחיל את המשחק

רפלקציה

במהלך העבודה הזו הרגשתי שאני לומד המון על השפה, ומתחיל להבין יותר ויותר איך המחשב עובד. לפי דעתי, שפת אסמבלי זו שפה נורא חשובה אפשר להרגיש שליטה מלאה על המחשב. בגלל רמת הlow-level שלה, אפשר לשנות

דברים במחשב כמו שאי אפשר לשנות בשום שפה אחרת.
העבודה הייתה נורא מהנה כי היא מראה מה אני למדתי במהלך השנה הזו, כאשר למדנו בבית הספר את השפה וגם כאשר למדתי בבית וניסיתי להתעמק יותר כדי להבין איך להשתמש בשפה הזו.

הספר האינטרנטי שנמצא באתר http://cyber.org.il/ של ברק גונן, שמלמד אסמבלי, עזר לי לשפר את היכולת שלי ללמוד לבד, ואת הידע שלי בשפה גם מחוץ ללימודים בכיתה.

הספר שביקשו מאתנו לכתוב לעבודה הזו, עזר לי להבין יותר את הקוד שלי ולראות איך רושמים ספר על פרויקט, שזה כלי שיוכל לשמש אותי לפרויקטים הבאים שאעשה.

הקוד המלא

```
.486
.model flat, stdcall
option casemap :none

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\msvcrt.inc
includelib msvcrt.lib

include drd.inc
includelib drd.lib

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


.data

msg db "Press OK if your ready to have fun!" , 0
cpt db "WARNING" , 0

is_arrow DWORD 0

robin_y DWORD 0
robin_x DWORD 0

arrow_y DWORD 0
arrow_x DWORD 0

rock_y DWORD 0
rock_x DWORD 0

is_fired DWORD 0

is_mon2_dead DWORD 0 ; when its 1, we know that the monster is dead

is_mon2_fired DWORD 0

mon1_y DWORD 520
mon1_x DWORD 0

mon2_y DWORD 150
mon2_x DWORD 712

mon3_y DWORD 325
mon3_x DWORD 700

mon3Bool1 DWORD 0
mon3Bool2 DWORD 0

mon2Bool1 DWORD 0
mon2Bool2 DWORD 0
```

```
firstRichAlive DWORD 1
secRichAlive DWORD 1
thirRichAlive DWORD 1

boolean_is_rock_no_need BYTE 0

mon1Way DWORD 1
mon2Way DWORD 1

GameSound db "Desmeon_-_Hellcat_[NCS_Release_.wav",0
LoseSound db "Frank_Sinatra_My_Way_With_Lyrics.wav",0
winSound db "Queen_-_We_are_the_champions_Chorus_only_.wav",0

changeWay DWORD 1

is_robin_win BYTE 0
is_robin_lose BYTE 0

imgDelMon1 BYTE 0
imgDelMon2 BYTE 0
imgDelMon3 BYTE 0
imgDelRock1 BYTE 0


aBg BYTE "forest.bmp",0
bg Img<0,0,0,0>

aRobin BYTE "robin.bmp",0
robin Img<0,0,0,0>

aGold BYTE "gold.bmp",0
gold Img<0,0,0,0>

aMonster1 BYTE "monster1.bmp",0
monster1 Img<0,0,0,0>

aMonster2 BYTE "monster2.bmp",0
monster2 Img<0,0,0,0>

aMonster3 BYTE "monster3.bmp",0
monster3 Img<0,0,0,0>

aArrow BYTE "arrow.bmp",0
arrow Img<0,0,0,0>

aRock BYTE "rock.bmp",0
rock Img<0,0,0,0>

aOpen BYTE "OpenningPic.bmp",0
open Img<0,0,0,0>

aRules BYTE "rules.bmp",0
rules Img<0,0,0,0>

aWin BYTE "WIN.bmp",0
win Img<0,0,0,0>
```

```
aLose BYTE "lose.bmp",0
lose Img<0,0,0,0>

.code

X macro args:VARARG
       asm_txt TEXTEQU <>
       FORC char,<&args>
              IFDIF <&char>,<!\>
                     asm_txt CATSTR asm_txt,<&char>
              ELSE
                     asm_txt
                     asm_txt TEXTEQU <>
              ENDIF
       ENDM
       asm_txt
endm

moveRobin PROC
X      invoke GetAsyncKeyState, VK_RIGHT \ cmp eax, 0 \ jne right

X      invoke GetAsyncKeyState, VK_LEFT \ cmp eax, 0 \ jne left

X      invoke GetAsyncKeyState, VK_UP \ cmp eax, 0 \ jne up

X      invoke GetAsyncKeyState, VK_DOWN \ cmp eax, 0 \ jne down

X      invoke GetAsyncKeyState, VK_SPACE \ cmp eax, 0 \ jne shoot

              jmp nothing

       right:
              cmp robin_x,725
              je nothing
              inc robin_x
              ret

       left:
              cmp robin_x,0
              je nothing
              dec robin_x
              ret

       up:
              cmp robin_y,0
              je nothing
              dec robin_y
              ret

       down:
              cmp robin_y,490
              je nothing
              inc robin_y
              ret


       shoot:
              mov is_arrow, 1
```

```
            ret

        nothing:
            ret

moveRobin ENDP


checkRockAroow PROC

        mov ecx , arrow_y
        mov eax , arrow_x

        mov ebx , rock_x
        cmp ebx , eax
        je checkNextX

        jmp nothing

        checkNextX:
            add eax , 62
            cmp ebx , eax
            jbe checkFirstY
            ret

        checkFirstY:
            mov ebx , rock_y
            cmp ecx , ebx
            jbe checkSecY
            ret

        checkSecY:
            add ecx , 20 ; for comparing the y of robins
            cmp ecx , ebx
            jge loseArrowAndRock
            ret

        loseArrowAndRock:
            mov boolean_is_rock_no_need , 1
            mov is_arrow, 0
            mov is_fired, 0
            ret

        nothing:
            ret

checkRockAroow ENDP


monster2Shooting PROC

        cmp is_mon2_fired, 1
        je drawRock

        cmp boolean_is_rock_no_need , 1
        je drawThisRock

        mov eax, mon2_x
```

```
        mov ebx, mon2_y

        mov is_mon2_fired, 1

        mov rock_x, eax
        mov rock_y, ebx

        add rock_y , 55
        sub rock_x , 88

        jmp nothing

        drawRock:
                cmp rock_x, 0
                je destroy_Rock

                invoke checkRockAroow

                dec rock_x
                invoke drd_imageDraw, offset rock, rock_x, rock_y
                ret

                drawThisRock:
                        ret

        destroy_Rock:
                mov is_mon2_fired, 0
                ret

        nothing:
                ret

monster2Shooting ENDP


moveMonster1 PROC
        cmp mon1Way , 1
        jne secMov1
        cmp mon1_y , 0
        je nothing1
        inc mon1_x
        dec mon1_y
        ret

        secMov1:
                cmp mon1Way , 2
                jne thrMon1
                cmp mon1_x , 720
                jae nothing1
                inc mon1_x
                inc mon1_y
                ret

        thrMon1:
                cmp mon1Way , 3
                jne final1
                cmp mon1_y , 520
                jae nothing1
```

```
            dec mon1_x
            inc mon1_y
            ret

      final1:
            cmp mon1_x , 0
            je endd1
            dec mon1_x
            cmp mon1_x , 0
            ret

      nothing1:
            inc mon1Way
            ret

      endd1:
            mov mon1Way , 1
            ret

moveMonster1 ENDP



moveMonster2 PROC
      mov eax , mon2Bool1
      mov ebx , mon2Bool2
      cmp eax , ebx
      je mon2switch1
      jne mon2switch3

      mon2switch1:
            cmp mon2_x , 350
            je mon2switch2
            dec mon2_x
            ret

      mon2switch2:
            cmp mon2_y , 489
            je incMon2Bool1
            inc mon2_y
            ret

      incMon2Bool1:
            inc mon2Bool1
            ret

      mon2switch3:
            cmp mon2_y , 150
            je mon2switch4
            dec mon2_y
            ret

      mon2switch4:
            cmp mon2_x , 712
            je incMon2Bool2
            inc mon2_x
            ret
```

```
        incMon2Bool2:
                inc mon2Bool2
                ret

moveMonster2 ENDP


moveMonster3 PROC
        mov eax , mon3Bool1
        mov ebx , mon3Bool2
        cmp eax , ebx
        je switch1
        jne switch3


        switch1:
                cmp mon3_x , 500
                je switch2
                dec mon3_x
                ret

        switch2:
                cmp mon3_y , 502
                je incMon3Bool1
                inc mon3_y
                ret

        incMon3Bool1:
                inc mon3Bool1
                ret

        switch3:
                cmp mon3_y , 325
                je switch4
                dec mon3_y
                ret

        switch4:
                cmp mon3_x , 700
                je incMon3Bool2
                inc mon3_x
                ret

        incMon3Bool2:
                inc mon3Bool2
                ret
moveMonster3 ENDP



checkIfWin PROC

        cmp robin_x , 725
        je checkFirstY

        jmp nothing

        checkFirstY:
```

```
            cmp robin_y , 415
            jge checkSecY
            ret

      checkSecY:
            cmp robin_y , 490
            jle jIfWin
            ret

      jIfWin:
            mov is_robin_win , 1
            ret

      nothing:
            ret

checkIfWin ENDP


hitMonster3 PROC

      mov ecx , robin_y
      mov eax , robin_x

      mov ebx , mon3_x
      cmp ebx , eax
      jge mon3CheckSecx

      jmp nothing

      mon3CheckSecx:
            add eax , 75
            cmp ebx , eax
            jbe mon3CheckFirstY
            ret

      mon3CheckFirstY:
            mov ebx , mon3_y
            cmp ecx , ebx
            jbe mon3CheckSecY
            ret

      mon3CheckSecY:
            add ecx , 110 ; for comparing the y of robins
            cmp ecx , ebx
            jge youLose
            ret

      youLose:
            mov is_robin_lose , 1
            ret

      nothing:
            ret

hitMonster3 ENDP
```

```
hitMonster1 PROC

        mov ecx , robin_y
        mov eax , robin_x

        mov ebx , mon1_x
        cmp ebx , eax
        jge mon1CheckSecx

        jmp nothing

        mon1CheckSecx:
                add eax , 75
                cmp ebx , eax
                jbe mon1CheckFirstY
                ret

        mon1CheckFirstY:
                mov ebx , mon1_y
                cmp ecx , ebx
                jbe mon1CheckSecY
                ret

        mon1CheckSecY:
                add ecx , 110 ; for comparing the y of robins
                cmp ecx , ebx
                jge youLose
                ret

        youLose:
                mov is_robin_lose , 1
                ret

        nothing:
                ret

hitMonster1 ENDP


hitMonster2 PROC

        mov ecx , robin_y
        mov eax , robin_x

        mov ebx , mon2_x
        cmp ebx , eax
        je mon2CheckSecx

        mon2CheckSecx:
                add eax , 75
                cmp ebx , eax
                jbe mon2CheckFirstY
                ret

        mon2CheckFirstY:
                mov ebx , mon2_y
                cmp ecx , ebx
                jbe mon2CheckSecY
```

```
            ret

        mon2CheckSecY:
            add ecx , 110 ; for comparing the y of robins
            cmp ecx , ebx
            jge youLose
            ret

        youLose:
            mov is_robin_lose , 1
            ret

        nothing:
            ret

hitMonster2 ENDP


hitRock PROC

        mov ecx , robin_y
        mov eax , robin_x

        mov ebx , rock_x
        cmp ebx , eax
        jge checkRockSecx

        jmp nothing

        checkRockSecx:
            add eax , 75
            cmp ebx , eax
            jbe checkRockFirstY
            ret

        checkRockFirstY:
            mov ebx , rock_y
            cmp ecx , ebx
            jbe checkRockSecY
            ret

        checkRockSecY:
            add ecx , 110 ; for comparing the y of robins
            cmp ecx , ebx
            jge youLose
            ret

        youLose:
            mov is_robin_lose , 1
            ret

        nothing:
            ret

hitRock ENDP


checkIfMon1Dead PROC
```

```
        cmp firstRichAlive, 1
        jne movFirstRich

        ret

        movFirstRich:

                cmp imgDelMon1 , 1
                je noDelAgain1

                invoke drd_imageDelete, offset monster1

                mov imgDelMon1 , 1
                noDelAgain1:

                mov mon1_x , 732
                mov mon1_y , 0

                ret
checkIfMon1Dead ENDP


checkIfMon2Dead PROC

        cmp secRichAlive, 1
        jne movSecRich

        ret

        movSecRich:

                cmp imgDelMon2 , 1
                je noDelAgain2

                invoke drd_imageDelete, offset monster2
                mov imgDelMon2 , 1

                noDelAgain2:

                mov boolean_is_rock_no_need , 1
                mov is_mon2_dead , 1
                mov mon2_x , 712
                mov mon2_y , 0
                ret
checkIfMon2Dead ENDP


checkIfMon3Dead PROC

        cmp thirRichAlive, 1
        jne movThrRich

        ret

        movThrRich:

                cmp imgDelMon3 , 1
```

```
            je noDelAgain3

            invoke drd_imageDelete, offset monster3
            mov imgDelMon3 , 1

            noDelAgain3:

            mov mon3_x , 700
            mov mon3_y , 0
            ret
checkIfMon3Dead ENDP


checkIfRockDead PROC

      cmp boolean_is_rock_no_need , 1
      je deleteRock

      ret

      deleteRock:
            cmp imgDelRock1 , 1
            je noDelAgainRock

            invoke drd_imageDelete, offset rock

            mov imgDelRock1 , 1

            noDelAgainRock:

            mov  is_mon2_fired , 0
            mov rock_x , 775
            mov rock_y , 0

            ret
checkIfRockDead ENDP


monster1Arrow PROC

      mov ecx , arrow_y
      mov eax , arrow_x
      add eax , 62

      mov ebx , mon1_x
      cmp ebx , eax
      je mon1checkY

      jmp nothing

      mon1checkY:
            mov ebx , mon1_y
            cmp ecx , ebx
            jge mon1checkSecY
            ret

      mon1checkSecY:
            add ebx , 69 ; for comparing the y of monster
```

```
            cmp ecx , ebx
            jbe destroyMon1
            ret

      destroyMon1:
            mov firstRichAlive , 0
            jmp destroy_arrow
            ret

      destroy_arrow:
            mov is_arrow, 0
            mov is_fired, 0
            ret

      nothing:
            ret

monster1Arrow ENDP

monster2Arrow PROC
      mov ecx , arrow_y
      mov eax , arrow_x
      add eax , 62

      mov ebx , mon2_x
      cmp ebx , eax
      je mon2checkY

      jmp nothing
      mon2checkY:
            mov ebx , mon2_y
            cmp ecx , ebx
            jge mon2checkSecY
            ret
      mon2checkSecY:
            add ebx , 111 ; for comparing the y of monster
            cmp ecx , ebx
            jbe destroyMon2
            ret
      destroyMon2:
            mov secRichAlive , 0
            mov is_mon2_dead , 1
            jmp destroy_arrow
            ret
      destroy_arrow:
            mov is_arrow, 0
            mov is_fired, 0
            ret

      nothing:
            ret
monster2Arrow ENDP


monster3Arrow PROC

      mov ecx , arrow_y
      mov eax , arrow_x
```

```
        add eax , 62

        mov ebx , mon3_x
        cmp ebx , eax
        je mon3checkY

        jmp nothing

        mon3checkY:
                mov ebx , mon3_y
                cmp ecx , ebx
                jge mon3checkSecY
                ret

        mon3checkSecY:
                add ebx , 98 ; for comparing the y of monster
                cmp ecx , ebx
                jbe destroyMon3
                ret

        destroyMon3:
                mov thirRichAlive , 0
                jmp destroy_arrow
                ret

        destroy_arrow:
                mov is_arrow, 0
                mov is_fired, 0
                ret

        nothing:
                ret

monster3Arrow ENDP

Shoot PROC

        cmp is_arrow, 0
        je destroy_arrow

        cmp is_fired, 1
        je draw

        mov eax, robin_x
        mov ebx, robin_y

        mov is_fired, 1

        mov arrow_x, eax
        mov arrow_y, ebx

        add arrow_y , 55
        add arrow_x , 75

        draw:
                cmp arrow_x, 737
                jge destroy_arrow
```

```
            mov ebx ,arrow_x
            add ebx , 62

            invoke monster1Arrow

            invoke monster2Arrow

            invoke monster3Arrow

            inc arrow_x
            invoke drd_imageDraw, offset arrow, arrow_x, arrow_y
            ret

            destroy_arrow:
            mov is_arrow, 0
            mov is_fired, 0
            ret

Shoot ENDP

makeSomething PROC



makeSomething ENDP


main PROC
       invoke MessageBox, NULL, addr msg , addr cpt, MB_OK
       invoke drd_init, 800, 600, INIT_WINDOW
       invoke PlaySound,addr GameSound,NULL,SND_ASYNC
       invoke drd_imageLoadFile,offset aBg, offset bg
       invoke drd_imageLoadFile,offset aRobin, offset robin
       invoke drd_imageLoadFile,offset aGold, offset gold
       invoke drd_imageLoadFile,offset aMonster1, offset monster1
       invoke drd_imageLoadFile,offset aMonster2, offset monster2
       invoke drd_imageLoadFile,offset aMonster3, offset monster3
       invoke drd_imageLoadFile,offset aArrow, offset arrow
       invoke drd_imageLoadFile,offset aRock, offset rock
       invoke drd_imageLoadFile,offset aOpen, offset open
       invoke drd_imageLoadFile,offset aRules, offset rules
       invoke drd_imageLoadFile,offset aWin, offset win
       invoke drd_imageLoadFile,offset aLose, offset lose

       invoke drd_imageSetTransparent, offset robin, 0FFFFFFh
       invoke drd_imageSetTransparent, offset gold, 0FFFFFFh
       invoke drd_imageSetTransparent, offset monster1, 0FFFFFFh
       invoke drd_imageSetTransparent, offset monster2, 0FFFFFFh
       invoke drd_imageSetTransparent, offset monster3, 0FFFFFFh
       invoke drd_imageSetTransparent, offset arrow, 0FFFFFFh
       invoke drd_imageSetTransparent, offset rock, 0FFFFFFh

       openGame:
              invoke drd_imageDraw , offset open , 0 , 0
              invoke drd_processMessages
              invoke drd_flip
              X      invoke GetAsyncKeyState, 52h \ cmp eax, 0 \ jne again
              X      invoke GetAsyncKeyState, 53h \ cmp eax, 0 \ jne rulesLoop
```

```
        jmp openGame

rulesLoop:
        invoke drd_imageDraw , offset rules , 0 , 0
        invoke drd_processMessages
        invoke drd_flip
        X       invoke GetAsyncKeyState, 52h \ cmp eax, 0 \ jne again
        X       invoke GetAsyncKeyState, 54h \ cmp eax, 0 \ jne openGame
        jmp rulesLoop

winLoop:
        invoke drd_imageDraw , offset win , 0 , 0
        invoke drd_processMessages
        invoke drd_flip
        jmp winLoop

loseLoop:
        invoke drd_imageDraw , offset lose , 0 , 0
        invoke drd_processMessages
        invoke drd_flip
        jmp loseLoop

playLoseSound:
        invoke PlaySound,NULL,NULL,SND_ASYNC
        invoke PlaySound,addr LoseSound,NULL,SND_ASYNC
        jmp loseLoop

playWinSound:
        invoke PlaySound,NULL,NULL,SND_ASYNC
        invoke PlaySound,addr winSound,NULL,SND_ASYNC
        jmp winLoop

again:
        invoke drd_imageDraw , offset bg , 0,0
        invoke Sleep,1

        ;Were making check if the character is dead
        cmp firstRichAlive, 1
        jne skip1Draw

        invoke moveMonster1
        invoke drd_imageDraw , offset monster1 , mon1_x , mon1_y
        skip1Draw:

        ;Were making check if the character is dead
        cmp secRichAlive , 1
        jne skip2Draw

        invoke moveMonster2
        invoke drd_imageDraw , offset monster2 , mon2_x , mon2_y
        skip2Draw:

        ;Were making check if the character is dead
        cmp thirRichAlive , 1
        jne skip3Draw

        invoke moveMonster3
        invoke drd_imageDraw , offset monster3 , mon3_x , mon3_y
```

```
            skip3Draw:

            invoke drd_imageDraw , offset robin , robin_x , robin_y
            invoke drd_imageDraw , offset gold, 725 , 490
            invoke drd_processMessages

            cmp is_mon2_dead , 1
            je skipMon2Shooting

            invoke monster2Shooting

            skipMon2Shooting:

            invoke moveRobin
            invoke Shoot
            invoke checkIfWin
            invoke hitMonster1
            invoke hitMonster2
            invoke hitMonster3
            invoke hitRock
            invoke checkIfMon1Dead
            invoke checkIfMon2Dead
            invoke checkIfMon3Dead
            invoke checkIfRockDead

            cmp is_robin_win , 1
            je playWinSound

            cmp is_robin_lose , 1
            je playLoseSound

            invoke drd_flip
            jmp again
            ret


main ENDP


end main
```