The Bubble Game

מגיש **פלג כדורי**
מנחה **יהודה אור**
18/06/2019






תוכן עניינים

מבוא ורפלקציה

פרוייקט זה הוא הפרוייקט המסכם של כיתה י' והוא מהווה 30 אחוזים מהבגרות במדעי המחשב. במהלך השנה למדנו לתכנת בשפה Assembly 86x MASM 32 בסביבת התכנות Visual Studio. במסגרת הפרוייקט, התבקשנו לכתוב משחק מחשב שיכלול את כל הידע שצברנו במהלך השנה.

הגעתי לשיעור עם ידע לא מועט במדעי המחשב אך לא באסמבלי. היה לי ניסיון תכנות בג'אווה אותו רכשתי במסגרת קבוצת הרובוטיקה ובמסגרת מספר פרוייקטים שכתבתי בקיץ. במחצית הראשונה לא הייתי בבית הספר, גרתי ולמדתי בסן פרנסיסקו. כשחזרתי לבית הספר הייתי צריך להשלים את כל החומר שנלמד ולהיות מוכן לכתוב פרוייקט. אני חושב שעמדתי במשימה זו בהצלחה.

במסגרת השיעורים ובמסגרת עצמית למדתי דברים רבים על טבעו של המחשב. למדתי על החלקים השונים בזכרון: על ההארד דרייב, הראם, המחסנית, האוגרים של ה cpu וה fpu. למדתי כיצד החלקים האלו מתקשרים ביניהם וכיצד המידע עובר ביניהם. בנוסף, למדתי הרבה על כתובות בראם ועל מצביעים, על כך שאפשר לגשת לערך של תא לפי הכתובת שלו ואפשר לשים בתאים כתובות של דברים אחרים וכך לארגן את הזיכרון. בנוסף, הבנתי אילו פעולות ה cpu יכול לבצע על מספרים ומה הם המגבלות שלו.
בנוסף לכך למדתי השנה איך ללמוד לבד ולהשתמש באינטרנט כדי למצוא תשובות לשאלות שלי. למדתי כיצד לכתוב קוד מסודר וגנרי וכיצד לדבג אותו באופן נכון.

כדי שאוכל להבין אסמבלי טוב יותר הורדתי ספר שמסביר על MASM 32 וקראתי חלקים ממנו כדי להעשיר את עצמי ולהיות מסוגל לכתוב קוד יותר טוב.

התחלתי את הפרוייקט רק בתור שומר מסך של בועות שיש לי על המחשב ואני מאוד אוהב. התחלתי לעבוד ולנסות לפתור את הבעיות בהן נתקלתי: חישוב התנגשות של בועות, שילוב מספרים עשרוניים בקוד ועוד בעיות רבות שצצו בהמשך. ככל שפתרתי את הבעיות רק נשאבתי יותר לתוך בפרוייקט ורציתי להוסיף דברים ולפתור עוד אתגרים. נהנתי מאוד לכתוב את הפרוייקט ואני בטוח שאמשיך לשפר אותו ולעבוד עליו גם לאחר שאגיש אותו.

המשחק

ניתן לשחק את המשחק כמה שחקנים שרוצים (כל עוד יש מספיק מקשים במקלדת). ניתן להגדיר את כמות השחקנים ומקשיהם בקובץ data.

מטרת המשחק:

להיות הדמות האחרונה שנשארת בחיים.

איך דמות מתה?

דמות מתה אם הניקוד שלה הוא 0 והיא מאבדת עוד נקודה או אם יש בועה אחת במשחק והדמות מתנגשת בה. דמות שמתה נעלמת מהמסך.

איך דמות מאבדת נקודה?

דמות מאבדת נקודה כאשר היא מתנגשת בבועה. כשזה קורה, הבועה בה הדמות התנגשה נעלמת.

איך דמות מרוויחה נקודה?

דמות מרוויחה נקודה כאשר היא מגיעה לתיבת הזהב. כשזה קורה, נוספת עוד בועה למשחק.

הוראות כלליות:

המשחק מתחיל כאשר כל הדמויות נמצאות בפינה הימנית התחתונה והניקוד של כל אחת מהן הוא 1

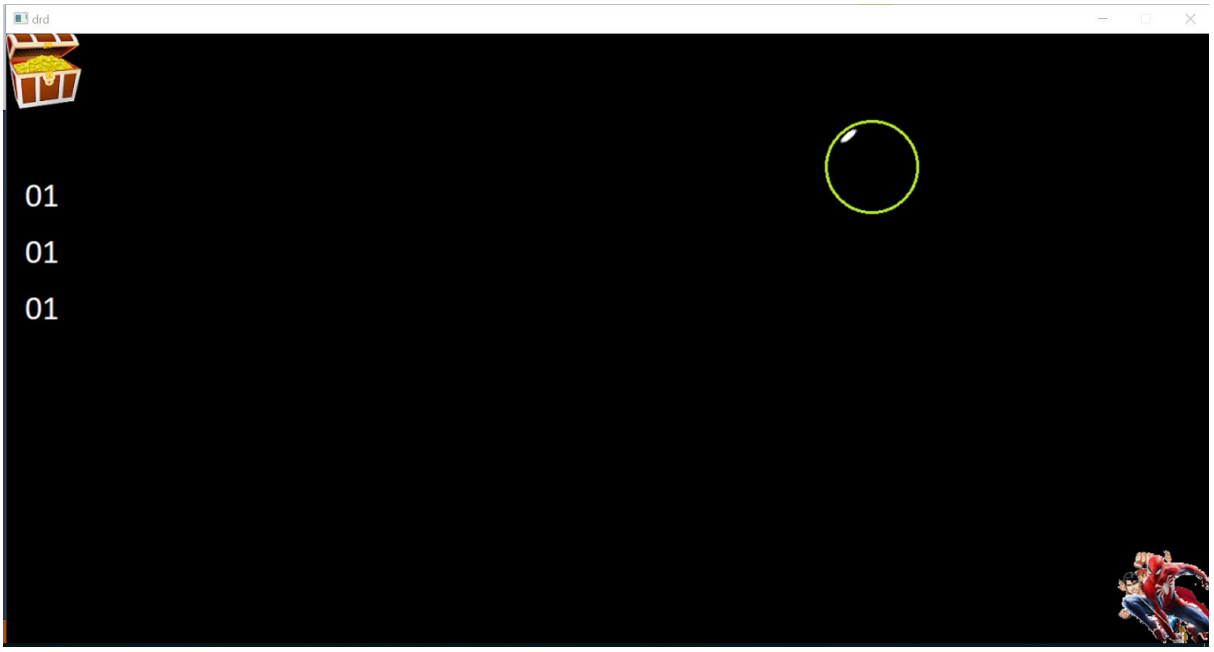
בכל פעם שדמות מרוויחה או מאבדת נקודה היא חוזרת לפינה הימנית התחתונה בה היא חסינה מהתנגשות למשך של כשתי שניות.

על כל דמות לנסות ולהרוויח כמה שיותר נקודות ולהוסיף בועות למשחק מבלי להיתקל בבועות. כך היא תרוויח נקודות, תקשה על היריבים שלה להרוויח נקודות ותביא לפסילתם.

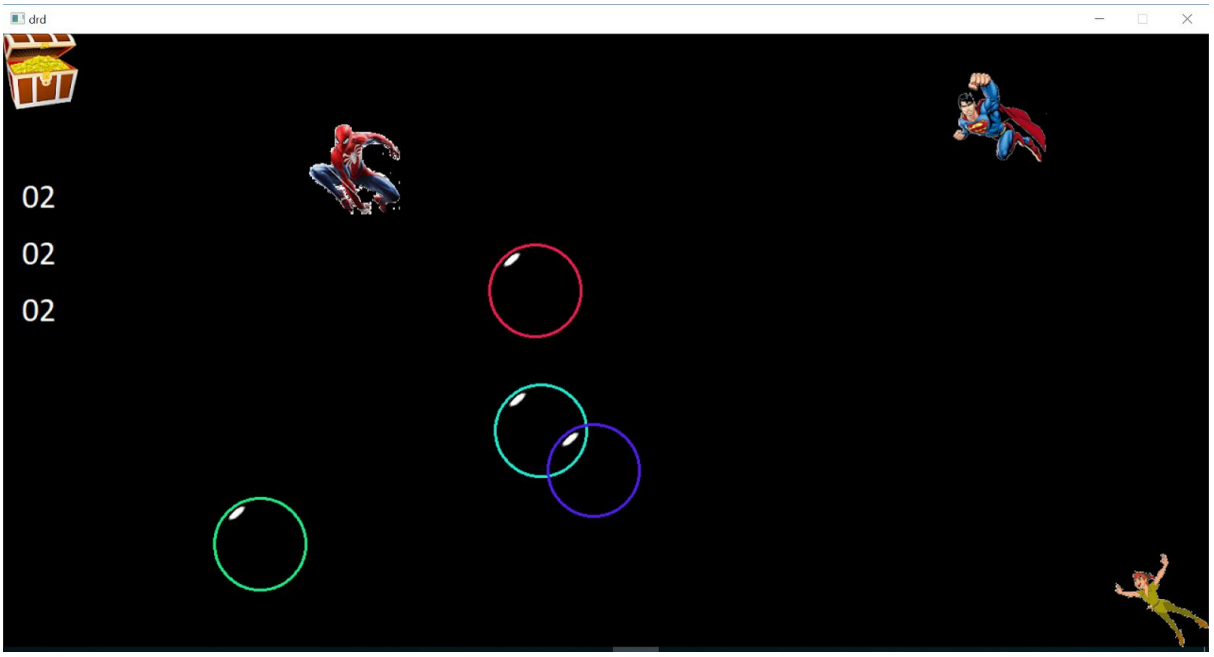
כאשר כל הדמויות נעלמות מהמסך המשחק הופך לשומר מסך של בועות כמו שומר המסך של Windows. בכמות הבועות ניתן גם לשלוט בקובץ data.

תמונות מהמשחק:

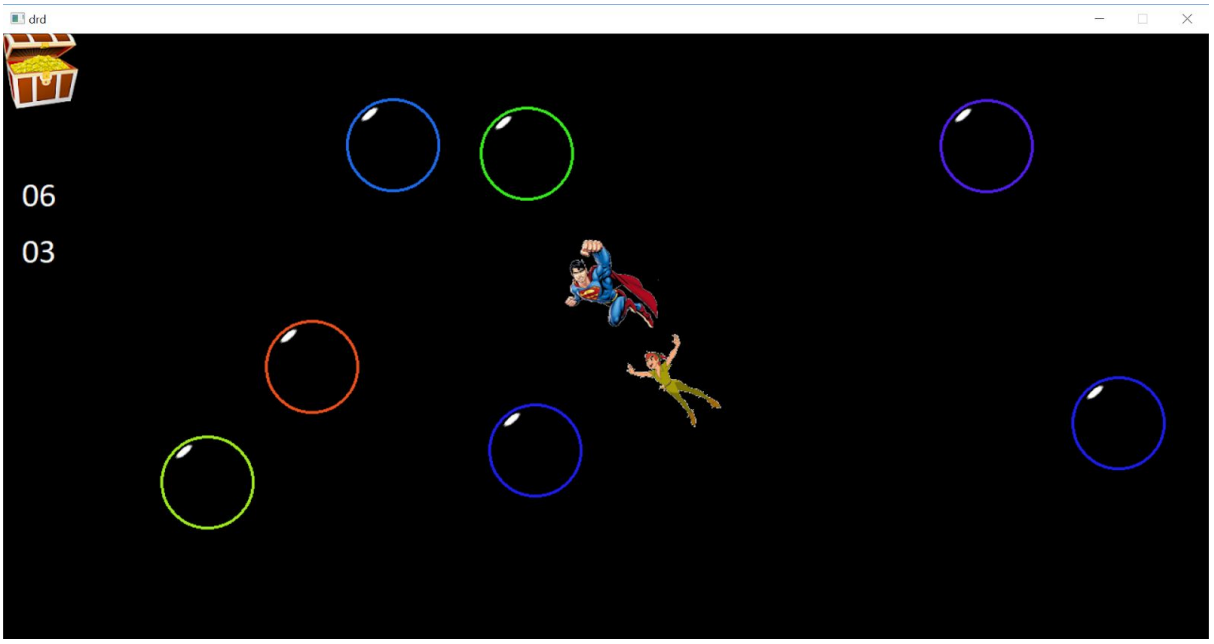
כך מתחיל המשחק עם שלוש דמויות



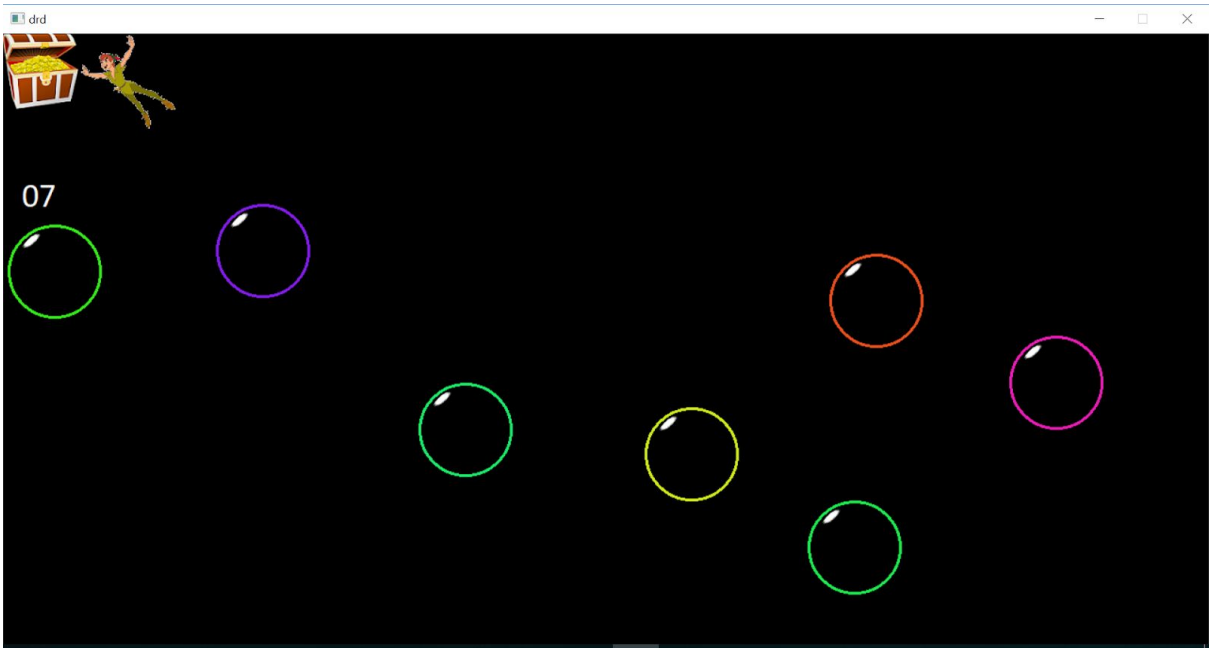
משחק בשלוש דמויות:



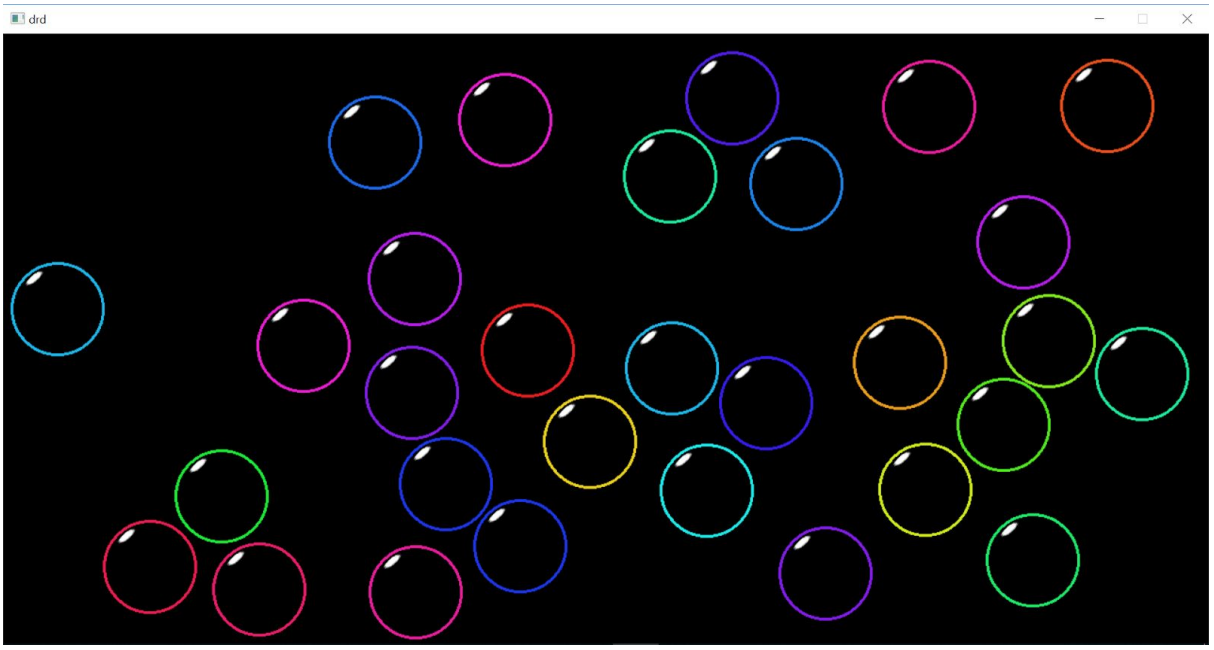
משחק בשתי דמויות:



כאשר דמות מנצחת:



שומר המסך לאחר שכל הדמויות מתו והמשחק נגמר:



חלק תאורטי

בשביל כתיבת הפרוייקט השתמשתי בידע באסמבלי שרכשתי במהלך השנה בשיעורים בבית הספר ובלמידה עצמאית בבית. כדי להרחיב את הידע שלי באסמבלי וללמוד עוד פקודות וללמוד כיצד דברים עובדים הורדתי את הספר Assembly Language for x86 Processors שכתב Kip Irvine . בעזרת הספר מצאתי בקלות כיצד יש להשתמש בפקודות נכון ועל ידי כך מנעתי טעויות רבות.

הצגת מספרים בhexadecimal:

ישנן הצגות נוספות למספרים מלבד ההצגה העשרונית. הצגה נוספת שמשתמשים בה רבות במדעי המחשב היא הצגה של מספרים בhexa. במקום להציג את המספרים בבסיס עשר מציגים אותם בבסיס 16 כאשר המספרים מ10 עד 15 הם a,b,c,d,e,f. למשל, השווי העשרוני של מספר כללי בן שתי ספרות xy הוא x*16 + y. למשל, ae = a*16 + e = 14 + 160 =174 .

Bit,Byte, Word, Dword

כל הזיכרון במחשב מורכב בסופו של דבר מביטים. הערך של bit יכול להיות רק 0 או 1. כדי לשמור מספרים על המחשב שומרים אותם בהצגה בינארית שמכילה רק 0 ו1. ההצגה הבינארית היא הצגה של מספר בבסיס 2. כך נראית ההצגה הבינארית של מספרים:

הקסהדצימאלי	בינארי	דצימאלי
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

דרושים 4 ביטים כדי להציג ספרה הקסדצימלית אחת.

Byte מורכב מ8 ביטים שהם 2 ספרות הקסדצימליות ותווך הערכים שלו הוא בין 0 ל255

הByte הוא היחידה הבסיסית ביותר של הזכרון. תא בזכרון מכיל byte אחד ואוסף של תאים יכול להכיל word, dword או כל מבנה נתונים אחר.

Word מורכב מ16 ביטים שהם 4 ספרות הקסדצימליות ותווך הערכים שלו הוא בין 0 ל65,535

Dword מורכב מ32 ביטים שהם 8ספרות הקסדצימליות ותווך הערכים שלו הוא בין 0 4,294,967,295

מבני נתונים:

ניתן להגדיר בקוד מבני נתונים (structs) המורכבים מתאים בזיכרון. ניתן ליצור משתנים של מבני הנתונים האלו ולאתחל אותם עם הערכים הרצויים. בקוד שלי השתמשתי במבני נתונים רבים שעזרו בארגון הקוד.

אוגרים(registers):

קיימים שמונה אוגרים כללים של 32bit:

eax, ebx, ecx, edx, esi, edi, ebp, esp

כאשר האוגרים esp,ebp שמורים לשימוש בstack בלבד.

ניתן להעביר ערכים מהזיכרון לאוגרים האילו, לבצע עליהם פעולות בעזרת הcpu ולהעבירם חזרה לזיכרון. האוגרים האלו מורכבים מאוגרים קטנים יותר בגדלים של word וbyte המאפשרים ביצוע פעולות גם על מספרים השמורים בתור word או byte. לאוגרים מסויימים ישנן תכונות מיוחדות שמאפשרות ביצוע של כל מיני פעולות רק עליהם, למשל את הפעולה mul ניתן לבצע רק על eax.

eax:edx הוא הרחבה של eax וedx לאוגר משותף של 64 ביט. משתמשים בו בפעולות החלוקה והכפל. ניתן להרחיב את התוכן של eax לתוך eax:edx על ידי הפעולה cdq.

הראם וההארד דרייב:

למחשב יש שני חלקים עיקריים בזיכרון. הRam והHard Drive. ההארד דרייב הוא הזכרון הקבוע של המחשב, הוא מכיל את כל המידע שנמצא על המחשב. כאשר המחשב מבצע תהליכים מסויימים הוא לא צריך את כל הנתונים בזיכרון שלו. לכן, ישנו את הראם שהוא זיכרון העבודה של המחשב. המחשב מעביר את הדברים שהוא רוצה לעבוד עליהם מההארד דרייב לראם ושם הוא יכול לגשת אליהם בקלות מפני שהראם קטן בהרבה מההארד דרייב וקל יותר לגשת למידע שעליו ולשנות אותו.

בפרוייקט העברתי את התמונות של השחקנים והבועות מההארד דרייב לראם בתחילת הקוד על ידי שימוש בפעולת הload.

כתובת של תא בראם:

בראם ישנם 2^32 תאים ממוספרים, כמו מספר הערכים בתווך המספרים של dword. לכן לכל תא בראם יש כתובת שניתן לגשת אליו לפיה. למשל אם יש תא בזכרון שהערך שלו הוא 5 ונסמן את הכתובת שלו בadr אז כדי לקבל את הערך של התא נצטרך לכתוב [adr].

כדי לקבל כתובת של תא בזכרון שקראנו לו ta יש להשתמש בפעולה offset ta. התוצאה שלה תהיה כתובת התא בזכרון.

flags:

אוסף של ביטים המתארים את המצב של המערכת. הם עוזרים לשורות בקוד לדעת מה קרה בשורות שלפניהן.

פקודות בהן השתמשתי לעיתים קרובות:

mov - פקודת mov היא אחת הפקודות הבסיסיות ביותר, היא מקבל שני פרמטרים בצורה הזו mov x,y ומעבירה את הערך של y אל x. x יכול להיות אוגר או תא בזכרון. משתמשים בה פעמים רבות כשרוצים להעביר מידע ממקום למקום במחשב.

add - הפקודה add x,y מוסיפה את y לx.

sub - הפקודה sub x,y מחסירה את y מx.

mul - פקודת mul x מכפילה את הערך של האוגר eax בx כאשר תוצאת המכפלה נמצאת ב eax:edx. בפקודה זו x חייב להיות אוגר אחר. mul מכפילה מספרים חיוביים וimul מכפילה גם מספרים שליליים.

div - הפקודה div x מחלקת את הערך של eax:edx בx ושמה את תוצאת החילוק בeax ואת השארית בedx. הפקודה div עובדת עבור מספרים חיוביים ואילו idiv עובדת גם עבור מספרים שליליים.

cmp - הפקודה cmp x,y משווה בין שני ערכים x,y היא משנה את הflags בהתאם.

jmp - כאשר מגדירים שורה בקוד בצורה הזו :NAME ניתן לעבור אליה משורה אחרת בקוד על ידי שימוש בפקודה jmp NAME.

conditional jumps - הפקודות קופצות לשורה במוגדרת אם הflags מתאימים לתנאי של הפקודה. נהוג להשתמש בפקודות האלו לאחר הפקודה cmp. הפקודה je NAME קופצת לשורה NAME אם הערכים שהשווינו אותם בcmp היו שווים. זו טבלה של כל הקפיצות לפי תנאי:

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

call - הפקודה call קופצת לשורה של הפונקציה ושומרת את המיקום בקוד ממנו קראנו לפונקציה על הstack מבצעת את הפעולות הרשומות שם ולאחר חוזרת אליו באמצעות הפעולה ret.

invoke - הפקודה invoke משתמשת בפעולה call ומאפשרת להעביר פרמטרים בקריאה לפונקצייה. היא מעלה את הפרמטרים לstack ומשתמשת בפקודה call. הפונקציה יכולה להשתמש בפרמטרים האלו.

מערכים:

מערך הוא אוסף של מבני נתונים בזכרון שבאים אחד אחרי השני. שימוש במערכים מאפשר גישה קלה למבני נתונים רבים ויצירה של כמות רבה של משתנים בשורת קוד אחת. ניתן להגדיר מערך באופן ידני כך: name dword 2,4,6 - יצירת מערך בין שלושה איברים מסוג dword שערכם 2,4,6 ניתן גם להגדיר מערך באמצעות שימוש בפקודה dup. דוגמה: name dword 36 dup (1) - יצירת מערך בן 36 איברים שערכם 1. dup משכפלת את מבנה הנתונים שבסוגריים לפי מספר הפעמים שהוגדר. כתובתו של מערך היא כתובת האיבר הראשון, כדי לגשת באיבר בו יש לחשב את הכתובת של האיבר לפי כתובת המערך, גודלו בזכרון של כל איבר במערכת, ומספר האיבר במערך.

ישנה גם דרך מקוצרת לגשת לאיבר במערך. כדי לקבל ערך של האיבר ה23 במערך שהגדרנו בעזרת הdup ניתן לכתוב name[4*22] כאשר 4 הוא גודלו של כל dword ו22 הוא האינדקס של האיבר ה23 במערך.

בפרוייקט שלי השתמשתי במערך כדי לתאר את כל הבועות שיצרתי. כדי ליצור את המערך הזה השתמשתי בdup.

FPU (floating point unit)

הFPU יכולה לעבוד עם שברים עשרוניים. הFPU מייצג שבר FPU כסכום של חזקות של חצי. 1:2, 1:4, 1:8, 1:16 ... בFPU מוגדרות מגוון פעולות מתמטיות מורכבות שלא קיימות ב CPU כגון: שורש,cos וsin. כשהתחלתי לעבוד על הפרוייקט לא ידעתי על קיומו של הFPU ומצאתי פתרון אחר לייצג מספרים עשרוניים. בפרוייקט הסופי השתמשתי בFPU רק פעם אחת כדי לחשב שורש.

חלק מעשי

תנועה והצגה גרפית:

התזוזה של הדמויות והבועות בפרוייקט נוצרת על ידי ציור של כל האובייקטים על המסך, הזזתם במעט וציור מחדש. התהליך הזה מתרחש במרווחי זמן זעירים ולכן נוצרת האשליה שהאובייקטים על המסך נעים בתנועה רציפה.

אתגרים בהם נתקלתי במהלך כתיבת הפרוייקט ופתרונם:

שברים עשרוניים:

שברים עשרוניים נחוצים ליצירת, זוויות לא שלמות, לתזוזה במהירויות מגוונות ולביצוע של פעולות מתמטיות מסויימות. לפי מה שלמדנו בכיתה אין מספרים עשרוניים ב CPU ובזיכרון. כדי להיות מסוגל לכתוב את הפרוייקט הזה הייתי חייב למצוא דרך לייצג שברים עשרוניים ולהיות מסוגל להתעסק איתם. מצאתי שיטה נוחה לעבוד עם שברים עשרוניים עוד לפני שלמדתי על קיומו של ה FPU.

השיטה שלי לייצוג של שברים עשרוניים מייצגת אותם באמצעות מספרים שלמים ומאפשרת דיוק של עד 4 ספרות אחרי הנקודה. המספר שמייצג כל שבר עשרוני הוא המכפלה של השבר העשרוני בקבוע שהגדרתי ב data (res.deci). בפרוייקט הקבוע הזה הוא 10,000. דוגמה: הייצוג של המספר 24.5675 הוא 245,675. על המספרים האלו הגדרתי פעולות כפל וחילוק שמשמרות על ההצגה הזו וכך הצלחתי לייצג שברים עשרוניים.

מספר המייצג שבר עשרוני נקרא מספר דצימלי (decimalic).

תזוזה במהירויות עשרוניות

גם כשהיו לי שברים עשרוניים לא יכולתי לאפשר תזוזה על המסך במהירויות ובזוויות שאינן שלמות מכיוון שהמיקום על המסך הוא מספר שלם. פתרתי את הבעיה הזו על ידי הוספת תכונה לכל אובייקט, כל אובייקט מכיל שלוש תכונות הקשורות במיקום ובמהירות:

1.pos - (position) מיקום האובייקט במספר רגיל

2.drcdec - (decimalic direction) וקטור המהירות של האובייקט בהצגה דצימלית

3.posdec - (decimalic direction) מיקום האובייקט בהצגה דצימלית

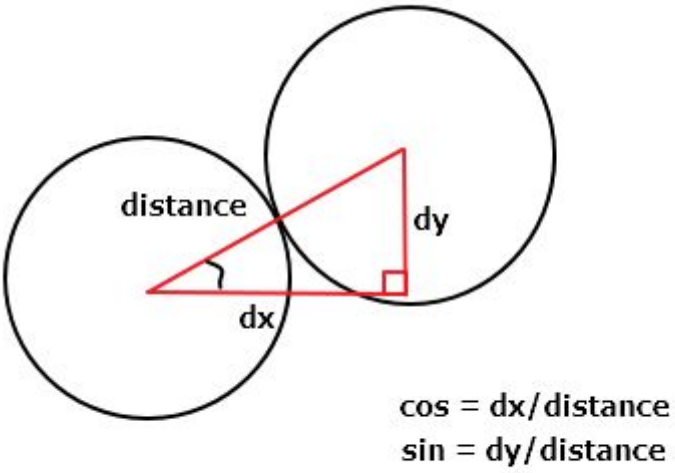
1 הוא המיקום של האובייקט על המסך ואילו 3 הוא המיקום המדויק של האובייקט. הפעולה שמעדכנת את המיקום של אובייקט updateloc מעדכנת את המיקום המדויק (3) על ידי הוספה של וקטור הכיוון המדויק 2. כדי לקבל את המיקום בהצגה רגילה שיוצג על המסך, הפעולה משתמשת בפעולת עזר round שמעגלת את המיקום המדויק למיקום בר הצגה. כך האובייקט יכול לזוז במהירויות ובזוויות שונות.

התנגשות בין שתי בועות

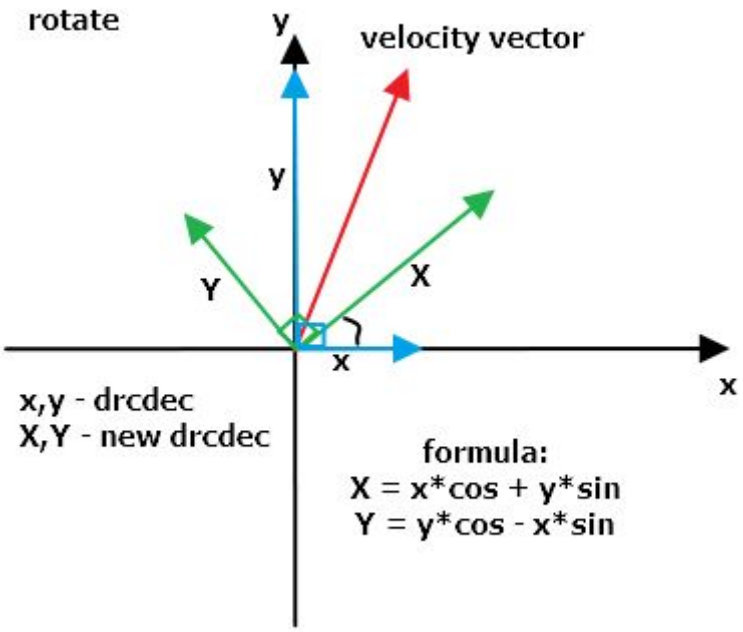
אתגר חשוב בפרוייקט היה למצוא נוסחה שתתאר התנגשות אלסטית בין שתי בועות ותקבע את ערכי המהירויות של הבועות לאחר ההתנגשות. למציאת הנוסחה הזו נעזרתי במשוואות שימור מומנט ואנרגיה, הנוסחה שמצאתי מתארת התנגשות אלסטית בין שתי בועות.

כדי לחשב את הנוסחה היה עליי למצוא את ערכי הקוסינוס והסינוס של זווית ההתנגשות. בהתחלה חשבתי להשתמש בטורי טיילור של קוסינוס, סינוס, וארכטנגנס שיעזרו לי לחשב את הזווית ומשם לחשב את ערכי הקוסינוס והסינוס.

מצאתי שיטה פשוטה יותר לחישוב ישיר של הקוסינוס והסינוס בה השתמשתי בפונקציה calangle. יצרתי משולש ישר זווית מהמרחק הכללי הפרש הx והפרש הy וחישבתי בעזרתו את ערכי הקוסינוס והסינוס כמוצג לעיל.



הנוסחה שמצאתי כדי לחשב את וקטורי המהירות של הבועות לאחר ההתנגשות היא להחליף בין הרכיבים הוקטורים של וקטורי המהירות הכלליים הנמצאים בכיוון זווית ההתנגשות. כדי לחשב את הרכיבים הנמצאים בזווית ההתנגשות ולהיות מסוגל להחליף ביניהם ולחשב את וקטור המהירות הכללי לאחר ההתנגשות כתבתי פעולה rotate. הפעולה מסובבת את הוקטורים במערכת לפי נוסחה שפיתחתי המשתמשת בקוסינוס ובסינוס שחישבנו.



הוקטור X הוא רכיב המהירות בזווית ההתנגשות אותו אני מחליף בין שתי הבועות ולאחר מכן אני מסובב את המערכת עם הזווית הנגדית לזווית ההתנגשות. כלומר, בחזרה למצבה הנורמלי. התוצאות שמתקבלות הן הx והy של וקטור המהירות הכללי של כל בועה. כלומר, drcdec.x ו drcdec.y המעודכנים שהם מה שהיינו צריכים (:

שימור אנרגייה

למרות הדיוק הרב שהשגתי בעזרת המספרים העשרוניים, גיליתי שהמערכת לא משמרת אנרגיה מסיבה כלשהי. לכן יצרתי שתי פעולות שעזרו לי לדבג ולהבין איפה הבעיה בקוד שלי. בעזרת הפעולה הראשונה שמחשבת את האנרגיה הכללית במערכת הבנתי שהאנרגיה יורדת בכל פעם שיש התנגשות בין בועות. הפעולה השנייה מחשבת את האנרגיה של זוג בועות נתונות. השתמשתי בה כדי לחשב את האנרגיה של זוג בועות בכל רגע נתון בזמן התנגשות. הפעולה אינה נמצאת בפרוייקט הסופי, כך היא הייתה כתובה:

```
;calculates the total energy of two bubbles and puts the result in energy
Energy proc adr1:dword,adr2:dword
        mov edi,adr1
        mov esi,adr2
        mov energy,0
        mov ebx,[edi + DM.drcdec.x]
        imul ebx,ebx
        mov ecx,[edi + DM.drcdec.y]
```

```
        imul ecx, ecx
        add ebx, ecx
        add energy,ebx
        mov ebx,[esi + DM.drcdec.x]
        imul ebx,ebx
        mov ecx,[esi + DM.drcdec.y]
        imul ecx, ecx
        add ebx, ecx
        add energy,ebx
ret
Energy endp
```

בעזרת הפעולה הבנתי שהבעיה הייתה שבחישוב הזווית הנחתי שהמרחק בין הבועות הוא המרחק שאותו הגדרתי כהתנגשות. לא הבנתי שיכול להיות שהמרחק הזה יכול להיות קטן בכמה אחוזים בגלל שבין הפריים הקודם לפריים הזה הבועות זזו "יותר מדי" וכעת הן אחת בתוך השנייה. כדי לפתור בעיה זו הייתי חייב לחשב את המרחק באופן מדויק וזה דרש למצוא דרך לחשב שורש של מספר. עד עכשיו כאשר השוויתי מרחקים השוותי בין ריבועי המרחקים לפי ובדקתי את התנאי הבא:

(dx)^2 + (dy)^2 <= distance^2

כדי לחשב שורש כתבתי פונקציה שמחשבת שורשים על ידי אלגוריתם לחילוץ שורש ממספר. הפונקצייה הזו עובדת והשתמשתי בה הרבה זמן בקוד אבל היא מניחה דברים מסוימים על המספרים שהיא מקבלת ובכך מונעת מהפרוייקט להיות גנרי. זו הסיבה שהעדפתי לעבור להשתמש ב FPU (שכבר ידעתי על קיומו ועל היכולות שלו) בשביל לחשב שורש. כך נראית הפונקצייה:

```
Root proc
        mov a,0 ;left num
        mov b,0 ;working num
        mov rc,0 ;result
        mov eax,dist
        mov dist,9

        cmp eax,10000
        je SCASE
        cdq
        mov ebx,100
        div ebx
        sub eax,81
        imul eax,100
```

```
        add eax,edx
        mov b,eax
        LOPP:
        mov eax,dist
        cdq
        mov ebx,10
        div ebx
        imul edx,2
        add edx,a
        imul edx,10
        mov a,edx
        mov ecx,0
        LOLP:
        mov ebx,a
        add ebx,ecx
        imul ebx,ecx
        cmp ebx,b
        jg ENDLOLP
        mov rc,ebx
        inc ecx
        cmp ecx,10
        jl LOLP
        ENDLOLP:
        dec ecx
        mov ebx,dist
        imul ebx,10
        add ebx,ecx
        mov dist,ebx
        cmp ebx,100000
        jg ROOTEND
        mov ebx,b
        sub ebx,rc
        imul ebx,100
        mov b,ebx
        jmp LOPP
        ROOTEND:
ret
SCASE:
mov dist,1000000
ret
Root endp
```

למרות המאמצים הרבים שהשקעתי בשימור האנרגיה, הדיוק שהשגתי בעזרת ארבע ספרות אחרי הנקודה לא הספיק ונותרה דליפה קטנה של אנרגיה. דליפה זו הפכה להיות נראית לעין רק לאחר זמן רב שהתוכנה פעלה. כדי לגבור על דליפה זו יצרתי פעולה שמחשבת את כל האנרגיה במערכת ומכפילה את המהירויות של הבועות כאשר האנרגיה עוברת סף תחתון שהגדרתי.
בנוסף, הגדרתי גם סף עליון לאנרגיה וקבוע בו מכפילים אם עוברים את הסף העליון. על ידי הורדת הערך של הסף העליון והפיכת קבוע ההכפלה שלו לגדול מאחד ניתן לגרום לאנרגיה לגדול עם הזמן. מצד שני על ידי העלאת הסף התחתון והפיכת הקבוע שלו לקטן מאחד ניתן לגרום לאנרגיה במערכת לדעוך.

כניסה של בועות

בהכנסת הבועות למסך נתקלתי במכשול נוסף, כאשר הכנסתי בועה וכבר הייתה בועה במקום בו הכנסתי נוצר באג, הקוד שכתבתי עד עכשיו לא ידע כיצד להתמודד עם זה. כדי לפתור בעיה זו נעזרתי ברעיון של windows והוספתי תכונת שקיפות לבועה. אם הבועה שקופה היא לא יכולה להתנגש באף בועה. הבועה נכנסת למסך שקופה ורק כאשר היא לא מתנגשת עם אף בועה אחרת היא נהיית מוצקה.

בועות שמתנגשות פעמיים

עוד בעיה שנוצרה כי לא לקחתי בחשבון את זה שבועות יכולות לחפוף אחת עם השנייה הייתה שבועות התנגשו זו בזו אבל בפריים הבא הן עדיין חפפו זו לזו והתנגשו בשנית. פריים לאחר מכן הן נעו במהירות המקורית שלהן זו לתוך זו והתנגשו בשלישית ולאחר מכן ברביעית ובחמישית... הבועות נשארו "דבוקות" זו לזו וגם הדביקו בועות אחרות. כדי לפתור את הבעיה הזו הוספתי לבועות תכונה lastcoladr. הבועות זוכרות את כתובת הבועה האחרונה איתן התנגשו (בהתחלה ובעת התנגשות עם קיר הכתובת הזו הופכת להיות הכתובת של הבועה עצמה). אם בהתנגשות שתי הבועות מצביעות זו על זו ההתנגשות מבוטלת וכך מנעתי את המצב הזה.

מגן באגים

למרות המאמצים הרבים שלי למנוע מצב בו הבועות "נדבקות" אחת לשנייה עדיין נותרו מקרי קצה נדירים שגרמו למצב הזה לקרות. למשל הבועות עלולות "להידבק", במקרים מסויימים של התנגשות משולשת או מרובעת באותו הפריים. כדי לפתור את מקרי הקצה יצרתי מגן באגים שבודק אם הבועות חופפות יותר מדי ואם כן הוא הופך אחת מהן לשקופה. כך הבועות הפסיקו להתנגש והבאג

נפתר. את המרחק המינימלי בו מגן הבאגים מחליט להפוך בועה ניתן לקבוע ב res.defensedist שנמצא בקובץ data.

יצירת אנימציה

יצירת האנימציה של הבועות היוותה אתגר נוסף. הייתי צריך למצוא דרך לטעון את כל התמונות של הבועות לראם באופן אוטומטי. לכן כתבתי פעולה המקבלת path אחד בהארד דרייב שיש לו אינדקס במקום ידוע ומערך של תמונות בגודל המתאים בראם. הפעולה יודעת החליף את האינדקס ב path ובכך להעלות את כל התמונות של הבועות למערך שלהן בראם בלולאה.

כדי לגרום לכך שכל אובייקט יחליף בין התמונות שלו ובכך תיווצר אנימציה יצרתי לכל אובייקט תכונה שהיא האינדקס של התמונה הנוכחית הוא מציג במערך התמונות של האובייקט. האינדקס הזה משתנה כל כמות מסוימת של פריימים המוגדרת במבנה המידע של האובייקט ולפיו משתנה התמונה של האובייקט.

התנגשות דמות בקיר

כאשר יצרתי את הדמויות הייתי חייב למצוא דרך למנוע מהדמות לצאת מגבולות המסך. כאשר הפעלתי את פעולת ההתנגשות בקיר של הבועות על הדמויות נוצרה בעיה. במקרים מסוימים כאשר הדמות נגעה בגבולות המסך המקשים שכיוונם אנכי לקיר החליפו כיוונים והדמות עדיין הצליחה לצאת מהמסך במצבים מסוימים.

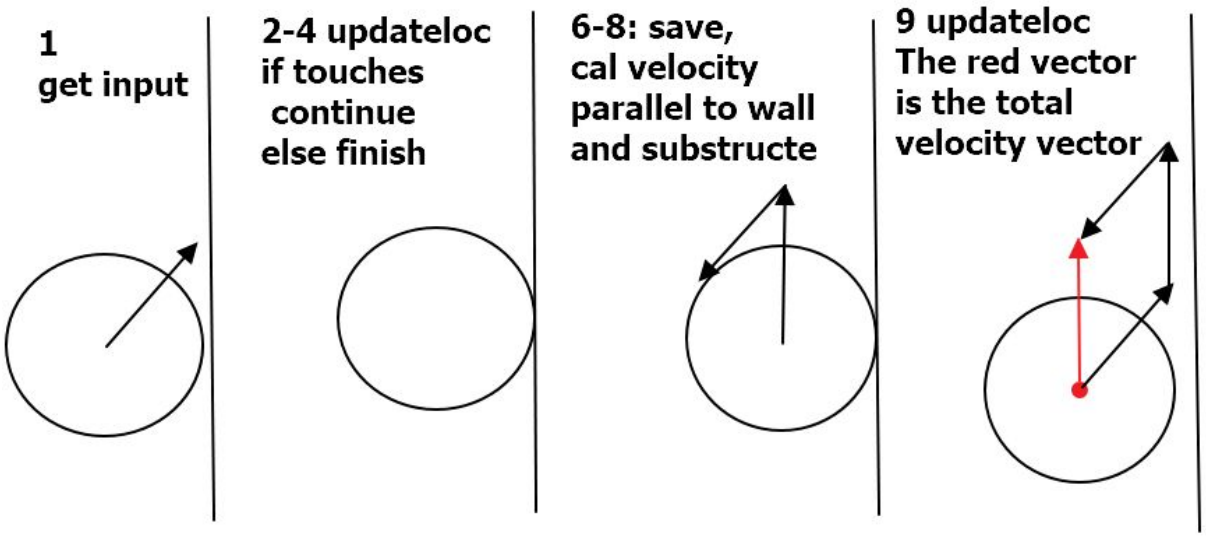
הפעולה שמטפלת במפגש של בועות עם הקיר הופכת את כיוון התנועה המאונך לקיר של הבועה. היא יודעת להבדיל רק בין מצב שזה קיר אנכי, אופקי או שניהם. היא מניחה שכיוון התנועה העכשווי של הבועה הוא כלפי הקיר כי רק כך הבועה יכולה להגיע אל הקיר. אצל הדמות לא ניתן להניח את ההנחה הזו, כיוון התנועה של הדמות יכול להשתנות בכל רגע נתון בהתאם למקשים עליהם לוחץ השחקן. לכן, כאשר הפעלתי את הפעולה הזו על הדמויות היא הפכה את סימני המקשים שכיוונם מאונך לקיר אבל לא מנעה מהדמות לצאת מגבולות המסך אם המקש שמאפשר זאת נלחץ.

עשיתי ניסיון לאפס את המהירות האנכית לקיר במקום להפוך את סימנה אבל ברגע שהדמות נגעה במסך היא נדבקה אליו כי מהירותה בכיוון האנכי לקיר בו נתקעה אופסה.

בסוף הבנתי שכדי לפתור את הבעיה בלי לכתוב פעולות חדשות מאפס עבור דמויות אני אצטרך לדאוג שאין מצב בו דמות נוגעת במסך בסוף פריים, כך אני יכול תמיד להניח שהדמות לא נוגעת בקיר כאשר מתחיל פריים חדש. הפתרון שלי עובד כך והוא ממומש בפעולה dmootmove בקוד:

אני מניח שהדמות לא נוגעת בקיר בתחילת הפריים

1. עדכון המהירות של הדמות לפי המקשים
2. עדכון המיקום הרגיל והדצימלי בעזרת הפעולה updateloc
3. בדיקה, האם הדמות מתנגשת בקיר
4. אם לא. סיימנו. בפריים הבא היא לא תתנגש בקיר לכן ההנחה תתקיים
5. אם כן:
6. שמירה של ערכי המהירות (drcdec).
7. עדכון המהירות על ידי איפוס הרכיב המאונך לקיר.
8. החסרת ערכי המהירות השמורים מערכי המהירות העדכניים.
9. עדכון המיקום.
10. בגלל שהחסרנו את ערכי המהירות השמורים בעצם ביטלנו את העדכון שעשינו בשלב 2. המהירות של הדמות תיקבע רק על פי המהירות שרכיבה האנכי לקיר הוא אפס ולכן הדמות לא תיגע בקיר גם בפריים הבא וההנחה תתקיים. התרשים הבא מסביר את שלבים 9 - 1:



הפיכת הקוד לגנרי

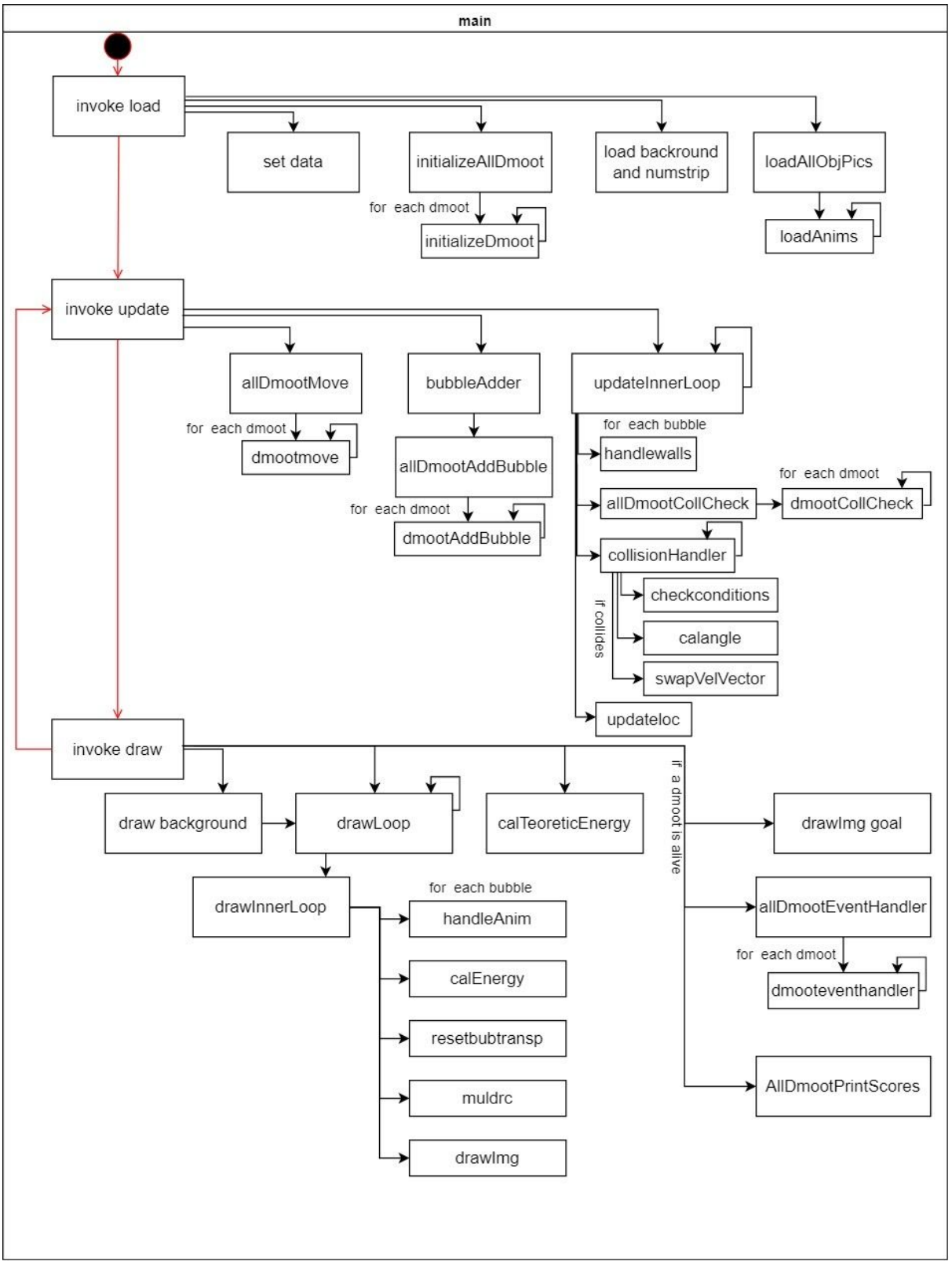
בהתחלה כשכתבתי את הקוד כתבתי אותו בצורה שתתאים רק למשחק הספציפי שאני תכננתי ותהיה מאוד קשה לשינוי. במהלך העבודה על הפרוייקט הבנתי שאם יהיה ניתן לעשות שינויים בקוד ובמשחק בקלות זה יפתח המון אפשריות לדברים שלא חשבתי עליהם. לכן השקעתי שעות רבות כדי להפוך את הקוד לגנרי. הוספתי מבני נתונים שמתארים כל סוג של אובייקט, הם מכילים את כל המידע הרלוונטי לאובייקט שנשאר קבוע במהלך המשחק, למשל: כתובת תמונה, רוחב תמונה, גובה תמונה, מספר אנימציות, מרחק התנגשות, מיקום התחלתי ומהירות, גודל הבית של דמות ועוד... כל

דמות מכילה את הכתובת של מבנה הנתונים שמתאר אותה וכך אפשר לגשת בקלות לכל הנתונים של דמות מסוימת בתוך הקוד בלי לדעת מי היא הדמות.

מבנה הנתונים Data מורכב ממבנה נתונים בסיסי ObjData הרלוונטי לגבי כל אובייקט, הוא מכיל מידע על התמונות ועוד נתונים בסיסיים על האובייקט. המבנה ObjData חייב להיות התכונה הראשונה במבנה Data מכיוון שכך אין צורך להבדיל בין אובייקטים שמבנה הנתונים שלהם הוא Data לאובייקטים שמבנה הנתונים שלהם הוא ObjData כאשר ניגשים לנתונים הבסיסיים שלהם הנמצאים בObjData. ניתן להניח שהכתובת שמצביעה למבנה הנתונים של האובייקט תמיד תצביע מבנה נתונים מסוג ObjData.
הדמויות במשחק מתוארות על ידי מבני נתונים מסוג Data ואילו הבועות מתוארות על ידי מבני נתונים מסוג ObjData.

בקובץ data הגדרתי שני מערכים שהופכים שינויים בקוד כמו הוספת דמות חדשה לגמרי למשחק (עם תמונות, אנימציה תכונות...) או יצירת דמות לפי מבנה Data קיים לעניין פשוט. אין צורך לשנות שום קובץ מלבד הוספת שורות ספורות בקובץ data. כדי ליצור דמות חדשה לגמרי יש ליצור מבנה נתונים Data חדש שיכיל את נתוני הדמות וליצור משתנה חדש של דמות המתוארת על ידי מבנה הנתונים הזה. לאחר מכן יש להוסיף את כתובת מבנה הנתונים מסוג Data שיצרנו למערך dataStructsOffsets ואת כתובת המשתנה של הדמות שיצרנו למערך offplayers התוכנה ניגשת לבד למערכים האלו ומבצעת את כל הפעולות כולל טעינת כל התמונות לראם על כל האיברים בהם.
לאחר שהפכתי את הקוד לגנרי ניתן לשלוט כמעט בכל דבר במשחק על ידי שינוי הקבועים שנמצאים בקובץ data.

תרשים UML


main
invoke load
set data
initializeAllDmoot
for each dmoot
initializeDmoot
load backround and numstrip
loadAllObjPics
loadAnims
invoke update
allDmootMove
for each dmoot
dmootmove
bubbleAdder
allDmootAddBubble
for each dmoot
dmootAddBubble
updateInnerLoop
for each bubble
handlewalls
allDmootCollCheck
for each dmoot
dmootCollCheck
collisionHandler
checkconditions
calangle
swapVelVector
if collides
updateloc
invoke draw
draw background
drawLoop
calTeoreticEnergy
drawInnerLoop
for each bubble
handleAnim
calEnergy
resetbubtransp
muldrc
drawImg
if a dmoot is alive
drawImg goal
allDmootEventHandler
for each dmoot
dmooteventhandler
AllDmootPrintScores

הקוד

data.inc

```
.const

;struct that contains the paths of the pictures in the hard drive
HD struct
	bubAnim BYTE "pics/bubble/im01.png",0
	piter BYTE "pics/piterpen.png",0
	bgpath byte "pics/bg.png",0
	numspath byte  "pics/nums.png",0
	superman byte "pics/superman.png",0
	goal byte "pics/goal.png",0
	spiderman byte "pics/spiderman.png",0
HD ends

;struct that contains Img structs in the ram. (the pictures)
Pics struct
	bubAnim Img 48 dup(<>)
	piter Img<>
	bg Img<>
	nums Img<>
	superman Img<>
	goal Img<>
	spiderman Img<>
Pics ends

; struct that contains the constants
Resorses struct
	;backround width
	wbg dword 1280

	;backround hight
	hbg dword 650

	;the decimal multiplication number
	deci dword 10000

	;how many runs the system needs to add a bubble
	addbubblecount dword 200
```

```
        ;energy conserve consts
        energyLowBound dword 9980
        energyLowConst     dword 10010
        energyHighBound dword 10020
        energyHighConst     dword 9990

        ;glich defense distance
        defensedist dword 90

        ;how many runs can a character stay transparanted at home
        hometimelimit dword 1000
        ;how many runs acharacter can't move after initialized
        freezeTime dword 300
Resorses ends

; struct of keys, defines the keys of each player
Keys struct
        left dword ?
        right dword ?
        up dword ?
        down dword ?
Keys ends

;struct defines a bubble - or any object that acts like a bubble
Bubble struct
        ;the position of the bubble in regular display
        pos POINT<>
        ;the direction & position of the bubble in decimal display
        drcdec POINT<>
        posdec POINT <>

        ;the offset of the Data struct describing the bubble
        data dword ?
        ;to match the Bubble struct with the Dmoot struct so DM struct would be
relevant
        ;and to be able to initialize the keys the Dmoot struct declaration statement
need to fill place in bubble
        nonsense dword 1,1,1,1;just filling in place (:

        ;contains the address of the last bubble to collide with this bubble
        ;in order to prevent bubbles from colliding many times in a row because they
still touch
```

```
	;when initialized or after a collision with the wall it contains the address of the
bubble itself
	lastcoladr dword 0

	;the bubble is transparent (it can't collide with other bubbles) if transp >=1
	transp dword 1 ;

	;the current picture index of the bubble
	animindex dword 0
Bubble ends

;struct describes characters (players)
Dmoot struct
	;same as bubble
	pos POINT<>
	drcdec POINT<>
	posdec POINT <>
	data dword ?

	;the keys of the player
	keys Keys<>

	lastcoladr dword 0

	;the character can't collide with bubbles if transp = 1
	transp dword 1

	;the current picture index of the character
	animindex dword 0

	;player's score if less than 0 player dies
	score dword 1

	;the number of runs that the player has spent at home (transparent) in a row
	;if greater than res.hometimelimit the player is not transparent anymore
	homeTime dword 0

	;the number of runs initializing the character to home that the char can't move
	freeze dword 0
Dmoot ends

;the initial position and direction data of a type of an object
```

```
ObjInit struct
	pos POINT<>
	drcdec POINT<>
	posdec POINT<>
ObjInit ends

;the data describes a general type of object
ObjData struct
	;an init struct - the program relies that this is the first item in the struct
	init ObjInit<>

	;picture hight and width
	imgh dword ?
	imgw dword ?

	;the collision distance of the object - mostly its frame parameter
	coldist dword ?

	;the location in the string of the picture path in the hd is the index of the
picture(if exists)
	imgconst dword ?

	;a constant for hitting a wall - multiplies the vertical velocity to the wall by it (-1
for bubble,0 for player)
	wallconst dword ?

	;the offset of the object picture/s in the ram and of the path in the hd
	offpic dword ?
	offpathd dword ?
	;transparanted color from the object's picture/s
	transpColor dword ?

	;number of pictures the object has
	animnum dword ?

	;number of runs for changing the picture with the next one
	animconst dword ?
ObjData ends

;struct defining a type of chracter
DmootData struct
	;general object struct - the program relies that this is the first item in the struct
```

```
        data ObjData <>
        ;character's decimal speed
        spdec dword ?
        ;size of a character's home x - width, y - hight
        homx dword ?
        homy dword ?
DmootData ends

;a struct that helps navigate between the Dmoot and bubble structs
;contains the place in the struct
DM struct
        pos POINT<0,4>
        drcdec POINT<8,12>
        posdec POINT<16,20>
        data dword 24
        keys Keys<28,32,36,40>
        lastcoladr dword 44
        transp dword 48
        animindex dword 52
        score dword 56
        hometime dword 60
        freeze dword 64
DM ends

;struct contains all the object types of the program - some values needs to be
initialized later
Data struct
        bubbled
ObjData<<<1,1>,<5000,5000>,<10000,10000>>,100,100,50,14,-1,?,?,0000000h,?,8
3>
        piterd
DmootData<<<<1180,550>,<0,0>,<11800000,5500000>>,100,100,20,1,0,?,?,0ffffffh,
?,1>,20000,200,200>
        supermand
DmootData<<<<1180,550>,<0,0>,<11800000,5500000>>,100,100,20,1,0,?,?,00000
00h,?,1>,20000,200,200>
        spidermand
DmootData<<<<1180,550>,<0,0>,<11800000,5500000>>,100,100,20,1,0,?,?,0ffffffh,
?,1>,20000,200,200>
        goald ObjData<<<0,0>,<0,0>,<0,0>>,80,80,30,1,-1,?,?,0000000h,?,1>
Data ends
```

```
;ram struct - contains the data and the pictures
Ram struct
        data Data<>
        pics Pics<>
Ram ends

.data
;create global variables
;uses to store results and save them during processes
result BYTE 0
;indexes of loops
i dword 0
j dword 0
sin dword 0
cos dword 0
;number of bubbles
n dword 1

;energy variables
energy dword 5000
TeoreticEnergy dword 0
dist dword 0

;create all the structs
res Resorses<>
hd HD<>
ram Ram<>

;array contains the addresses of all of the structs describing objects
dataStructsOffsets dword offset ram.data.bubbled,offset ram.data.piterd,offset
ram.data.supermand,offset ram.data.goald,offset ram.data.spidermand

;create players
piter Dmoot <<?,?>,<?,?>,<?,?>,offset
ram.data.piterd,<VK_LEFT,VK_RIGHT,VK_UP,VK_DOWN>>
piter2 Dmoot <<?,?>,<?,?>,<?,?>,offset
ram.data.piterd,<VK_A,VK_D,VK_W,VK_S>>
piter3 Dmoot <<?,?>,<?,?>,<?,?>,offset
ram.data.piterd,<VK_G,VK_J,VK_Y,VK_H>>
superman Dmoot<<?,?>,<?,?>,<?,?>,offset
ram.data.supermand,<VK_A,VK_D,VK_W,VK_S>>
```

```
spiderman Dmoot<<?,?>,<?,?>,<?,?>,offset
ram.data.spidermand,<VK_H,VK_K,VK_U,VK_J>>

;array contains player's addresses
offplayers dword offset piter,offset superman,offset spiderman
AlivePlayersNum dword LENGTHOF offplayers

;create bubbles - array using dup and a goal "bubble"
goal Bubble <<0,0>,<0,0>,<0,0>,offset ram.data.goald>
bubbles Bubble 30 dup(<<1,1>,<5000,5000>,<10000,10000>,offset
ram.data.bubbled>)

.code
;initialize the locations of the pictures in the ram and in the hd in the object structs
;and also the number of pictures for each object type
setData proc
        mov ram.data.bubbled.offpic,offset ram.pics.bubAnim
        mov ram.data.goald.offpic,offset ram.pics.goal
        mov ram.data.piterd.data.offpic,offset ram.pics.piter
        mov ram.data.supermand.data.offpic,offset ram.pics.superman
        mov ram.data.spidermand.data.offpic,offset ram.pics.spiderman

        mov ram.data.piterd.data.offpathd,offset hd.piter
        mov ram.data.supermand.data.offpathd,offset hd.superman
        mov ram.data.spidermand.data.offpathd,offset hd.spiderman
        mov ram.data.bubbled.offpathd,offset hd.bubAnim
        mov ram.data.goald.offpathd,offset hd.goal

        mov ram.data.bubbled.animnum,LENGTHOF ram.pics.bubAnim
        mov ram.data.goald.animnum,LENGTHOF ram.pics.goal
        mov ram.data.piterd.data.animnum,LENGTHOF ram.pics.piter
        mov ram.data.supermand.data.animnum,LENGTHOF ram.pics.superman
        mov ram.data.spidermand.data.animnum,LENGTHOF ram.pics.spiderman
ret
setData endp
```

Math_funcs.inc

```
.code

; definition of "decimalic number":
; integer which represents a decimal fraction, and is in the ten-thousands.
; the decimalic number for a decimal fraction P is given by: (res.deci)*P. (res.deci is a
constant)
; As example:
; res.deci = 10000, P=1.2567 ==> decimalic number = 12567
; In all following function comments, assume that res.deci = 10000.


; function receives a number, divisor, and performs a special kind of division.
; result = (number*(res.deci)) / divisor.
; num and divisor must be the same "kind"  of numbers.
; a. regular numbers
; b. decimalic numbers.
; the result is put in the address specified by adrret.


division proc num:dword, divisor:dword, adrret:dword
        mov eax,num
        mov ebx,res.deci
        imul ebx
        idiv divisor
        mov ebx,adrret
        mov [ebx],eax
ret
division endp

; receives a decimalic number (decnum) and rounds it.
; the result is put in the address specified by adrret.

round proc decnum:dword,adrret:dword
        mov eax, res.deci
        cdq
        mov ebx,2
        div ebx
        add eax,decnum ;until here eax = decnum + res.deci/2 = decnum + 500
        cdq
        idiv res.deci ;divied eax by res.deci, now eax is the rounded number
```

```
        mov ebx,adrret
        mov [ebx],eax
        ;put eax in the given address
ret
round endp

; multiplies the number in address "adr", by the decimalic number "muli" and divides
it by res.deci.
; the result is put in "adr"
; example: "adr" contains value 220, muli=9000, result= (220*9000)/res.deci = 198 =
90% of 220
muladr proc adr:dword,muli:dword
        mov ecx,adr
        mov eax,[ecx]
        mov ebx,muli
        imul ebx
        idiv res.deci
        mov [ecx],eax
ret
muladr endp

;copy a given section in the memory to another place in the memory
;receives the address where the section starts and it's length and copies each byte to
it's place at the target address
CopyMem proc adrstart:dword,Bytesnum:dword,adrtarget:dword
        mov ebx,adrstart
        mov ecx,adrtarget
        add Bytesnum,ebx
        LOOPcop:
        mov al,[ebx]
        mov [ecx],al
        add ebx,SIZEOF byte
        add ecx,SIZEOF byte
        cmp ebx,Bytesnum
        jl LOOPcop
ret
CopyMem endp
```

Func.inc

```
include Math_funcs.inc

; For definition of "decimalic", see Math_funcs.inc
; In this file, "character" usually refers to a character on the screen, and not a char.
.code

;checks if all the bubbles are in. if not increases n (which adds a bubble)
addBubble proc
	cmp n,LENGTHOF bubbles
	je RETADDBUBBLE
	inc n
	RETADDBUBBLE:
ret
addBubble endp

;checks if a character hits the walls return the state in result
;receives the character's address, width, and hight
;if it hits the vertical wall: result = 1. if the horizontal = 2. if both = 3. if none result = 0
;uses the position on the screen and not the decimalic position
checkwalls proc adrObj:dword, objwidth:dword, objhight:dword
	mov result,0
	mov ebx,adrObj
	mov ecx,objwidth
	mov edx,objhight
	mov eax,[ebx+DM.pos.x]
	cmp eax,0
	jng RES1
	add eax,ecx
	cmp eax,res.wbg
	jnl RES1
	RETRES1:
	mov eax,[ebx+DM.pos.y]
	cmp eax,0
	jng RES2
	add eax,edx
	cmp eax,res.hbg
	jnl RES2
ret

	RES1:
```

```
        add result,1
        jmp RETRES1

        RES2:
        add result,2
        ret

checkwalls endp

;checks collision between two characters and updates the result in result
;receives the character's addresses and the distance of the collision
;uses the position on the screen and not the decimalic position
;if the distance between the characters is lower or equal to the distance the result is
1 (true) else it is 0(false)
;the equation for comparing the distances is: (dx)^2 + (dy)^2 <= distance^2
;dx is the x difference between the characters.

collisioncheck proc adrObj:dword, adrObj2:dword, distance:dword
        mov eax,adrObj
        mov edx,adrObj2
        mov result,0
        mov ebx,[eax+DM.pos.x]
        mov ecx,[edx+DM.pos.x]
        sub ebx,[edx+DM.pos.x]
        imul ebx,ebx
        mov ecx,[eax+DM.pos.y]
        sub ecx,[edx+DM.pos.y]
        imul ecx,ecx
        add ebx,ecx
        mov ecx,distance
        imul ecx,ecx
        cmp ebx,ecx
        jg RESS0
        mov result,1
        RESS0:
ret
collisioncheck endp

;invokes collisioncheck with the two given addresses after calculating the default
collision distance of the Objects
;the total collision distance is the sum of each object's collision distance
collcheck proc adrObj:dword,adrObj2:dword
```

```
        mov ebx,adrObj
        mov ebx,[ebx + DM.data]
        mov ebx,(ObjData PTR [ebx]).coldist
        mov ecx,adrObj2
        mov ecx,[ecx + DM.data]
        add ebx,(ObjData PTR [ecx]).coldist
        invoke collisioncheck,adrObj,adrObj2,ebx
ret
collcheck endp

;receives a character's address and rotates its decimalic direction (drcdec) according
to the sin and cos of the angle
;after the rotation x becomes the length vector in the rotation angle, y becomes the
length vector in the rotation angle + 90
;follows the equations below when X is the new drcdec.x and x is the old one, same
for y
;X = x*cos + y*sin
;Y = y*cos - x*sin
;all the numbers are decimalic - that's why dividing by res.deci, make sure it stays
decimalic

rotate proc adrObj:dword
        mov ebx,adrObj
        mov ecx,[ebx + DM.drcdec.x]
        mov eax,[ebx + DM.drcdec.y]
        imul ecx, cos
        imul sin
        add eax,ecx
        cdq
        idiv res.deci
        mov ecx,[ebx + DM.drcdec.x]
        mov [ebx + DM.drcdec.x],eax
        mov eax,[ebx + DM.drcdec.y]
        imul ecx, sin
        imul cos
        sub eax,ecx
        cdq
        idiv res.deci
        mov [ebx + DM.drcdec.y],eax
ret
rotate endp
```

```
;updates the location of a character according to its direction
;receives the character's address as an input
;for both x and y it sums up the accurate position and direction(posdec,drcdec)
;it updates posdec to the new accurate location and rounds it to display it on the
screen
;the round function puts the rounded value in the character's (not decimalic) position

updateloc proc adrObj:dword
        mov eax,adrObj
        mov ebx,[eax+DM.posdec.x]
        add ebx,[eax+DM.drcdec.x]
        mov [eax+DM.posdec.x],ebx
        add eax,DM.pos.x
        invoke round,ebx, eax
        mov eax,adrObj
        mov ebx,[eax+DM.posdec.y]
        add ebx,[eax+DM.drcdec.y]
        mov [eax+DM.posdec.y],ebx
        add eax,DM.pos.y
        invoke round,ebx, eax
ret
updateloc endp

;if there is a collision this function is called to make sure both bubbles are solid
;receives a bubble's address and check its transp state
;if (transp == true) then it increases the result and changes transp state so the
bubble would remain transp
;the bubble shouldn't remain transp because it collided with another bubble

transphandler proc adrObj:dword
        mov eax,adrObj
        cmp [eax + DM.transp],1
        jl NOTHING
        inc result
        add [eax + DM.transp],1
        NOTHING:
ret
transphandler endp

;loads all the pictures of an object/ or just a bunch of pictures
;receives the addresses of the: first picture path, the pictures array in the ram,
```

```
;the color to be transparanted, and after how many chars in the path is the picture
index
;j is the loop index and i is the address of the current picture in the ram

loadanims proc
adrhd:dword,adram:dword,color:dword,animnum:dword,imgconst:dword
        mov j,0
        mov eax,adram
        mov i,eax
        LOADLOOP:
        ;foreach picture - load the picture and transparent the given color
        invoke drd_imageLoadFile,adrhd,i
        invoke drd_imageSetTransparent,i,color
        inc j
        add i,SIZEOF Img

        ;puts in al the tens digit and in ah the one's digit of j+1
        mov ax,WORD PTR j
        inc ax
        mov bl,10
        div bl

        ;converts to ascii char
        add al,48
        add ah,48

        ;movs the chars to the index place in path string
        mov ecx,adrhd
        add ecx,imgconst
        mov [ecx],al
        mov [ecx +1],ah

        ;checks if finished
        mov ecx,animnum
        cmp j,ecx
        jl LOADLOOP
ret
loadanims endp

;invokes loadanims for a type of object(Data struct)
loadAnims proc adrdata:dword
        mov ebx,adrdata
```

```
        invoke loadanims,(ObjData PTR [ebx]).offpathd,(ObjData PTR
[ebx]).offpic,(ObjData PTR [ebx]).transpColor,(ObjData PTR
[ebx]).animnum,(ObjData PTR [ebx]).imgconst
ret
loadAnims endp

;load all anims foreach type of object in a given array
loadAllObjPics proc adrdata:dword,num:dword
        AGAIN:
        cmp num,0
        je ENDLOAD
        mov ebx,adrdata
        invoke loadAnims,[ebx]
        add adrdata,SIZEOF dword
        dec num
        jmp AGAIN
        ENDLOAD:
ret
loadAllObjPics endp

;multiplies a direction aith a given decimal factor of an object using muladr proc
muldrc proc adrdm:dword,factordec:dword
        mov ebx,adrdm
        add ebx,DM.drcdec.x
        invoke muladr,ebx,factordec
        mov ebx,adrdm
        add ebx,DM.drcdec.y
        invoke muladr,ebx,factordec
ret
muldrc endp

;updates the anim index of an object corresponding to the counter value (how many
runs)
;and the animconst specified in the type of object Data struct
handleAnim proc adrObj:dword,counter:dword
        ;divides counter by animconst
        mov ecx,adrObj
        mov eax,counter
        cdq
        mov ebx,[ecx+DM.data]
        mov ebx,(ObjData PTR [ebx]).animconst
        div ebx
```

```
	;if remainder is 0 increase object's animindex and make sure it is valid (not
over the max animindex)
	cmp edx,0
	jne NOTCHANGEANIM
	mov ecx,adrObj
	mov eax,[ecx+DM.animindex]
	inc eax
	cdq
	mov ebx,[ecx+DM.data]
	mov ebx,(ObjData PTR [ebx]).animnum
	div ebx
	mov [ecx+DM.animindex],edx
	NOTCHANGEANIM:
ret
handleAnim endp

;receives an object's address, check for collision with walls and acts correspondingly
handlewalls proc adr:dword
	;check collision with checkwalls proc
	mov ebx,adr
	mov ecx,[ebx+DM.data]
	invoke checkwalls,adr,(ObjData PTR [ecx]).imgw,(ObjData PTR [ecx]).imgh
	mov ebx,adr
	mov ecx,[ebx+DM.data]
	mov edx,(ObjData PTR [ecx]).wallconst

	;result >= 2 if collided with horizontal wall
	cmp result,2
	jl NOTNEGY
	mov [ebx + DM.lastcoladr], ebx ;makes sure a collision won't be cancelled
	;muls y direction by the object's wall const
	mov ecx,[ebx+DM.drcdec.y]
	imul ecx,edx
	mov [ebx+DM.drcdec.y],ecx
	sub result, 2
	NOTNEGY:
	;result = 1 if  collided with vertical wall
	;does the same as y for x

	cmp result,1
	jne NOTNEGX
```

```
        mov [ebx + DM.lastcoladr], ebx ;makes sure a collision won't be cancelled
        mov ecx,[ebx+DM.drcdec.x]
        imul ecx,edx
        mov [ebx+DM.drcdec.x],ecx
        NOTNEGX:
ret
handlewalls endp

;calculate the theoretic energy in the system with the bubble's initial velocities
calTeoreticEnergy proc
        ;TeoreticEnergy = n*(vel x^2 + vel y ^2)
        mov ebx,ram.data.bubbled.init.drcdec.x
        imul ebx,ebx
        mov eax,ram.data.bubbled.init.drcdec.y
        mul eax
        add eax,ebx
        cdq
        idiv res.deci
        imul eax,n
        mov TeoreticEnergy,eax
ret
calTeoreticEnergy endp

;print a two digits number on the screen according to the input
printNum proc num:dword,posx:dword,posy:dword
        ;puts in eax the tens digit and in edx the one's digit of num
        mov eax,num
        mov ebx,10
        cdq
        div ebx
        ;converts them to the x pos of the digit in the strip
        imul edx,18
        mov num,edx ;saves edx in num
        imul eax,18
        ;draw the digits according to the data
        invoke drd_imageDrawCrop,offset ram.pics.nums,posx,posy,eax,0,18,40
        add posx,18
        invoke drd_imageDrawCrop,offset ram.pics.nums,posx,posy,num,0,18,40
ret
printNum endp

;draw an object's img acoording to it's data
```

```
drawImg proc adrObj:dword
        mov ecx,adrObj
        imul ebx,[ecx+DM.animindex],SIZEOF Img
        mov edx,[ecx + DM.data]
        add ebx, (ObjData PTR [edx]).offpic
        invoke drd_imageDraw, ebx,[ecx+DM.pos.x],[ecx+DM.pos.y]
ret
drawImg endp
```

Dmoot_funcs.inc

```
include Func.inc
.code

;receives player and a key, if the key is pressed, adds the value to the given place in
the direction
;add 1 to the result if key is pressed
actByKey proc key:dword, adrObj:dword, drc:dword,value:dword
	;checks if key is pressed
	mov ebx,adrObj
	add ebx,key
	invoke GetAsyncKeyState,[ebx]
	mov dl,1
	cmp eax,0
	je TOHERE
	mov dl,0
	;adds value to direction
	mov ecx,value
	mov ebx,adrObj
	add ebx,drc
	add [ebx], ecx
	TOHERE:
	add result,dl
ret
actByKey endp

;update a player's direction according to the keys' state and to the object's speed
;if two keys are pressed, decrease the direction so the total velocity would alway be
the same
keyboard proc adrObj:dword,factorspd:dword
	;drc,result =0
	mov ebx,adrObj
	mov [ebx+DM.drcdec.x],0
	mov [ebx+DM.drcdec.y],0
	mov result,0

	;checks the state of each key and updates drc using actbykey proc
	invoke actByKey, DM.keys.right, adrObj, DM.drcdec.x,res.deci
	invoke actByKey, DM.keys.down, adrObj, DM.drcdec.y,res.deci
	mov ebx,res.deci
	neg ebx
```

```
        invoke actByKey, DM.keys.up, adrObj, DM.drcdec.y,ebx
        mov ebx,res.deci
        neg ebx
        invoke actByKey, DM.keys.left, adrObj, DM.drcdec.x,ebx
        ;if two keys are pressed mul drc by 1/sqrt(2)
        cmp result,2
        jne FINALE
        invoke muldrc,adrObj,7071
        FINALE:
        ;mul by object's velocity
        invoke muldrc,adrObj,factorspd
        ret
keyboard endp

;updates the direction corresponding to the key's states and the location using
updateloc proc
;checks if collided with the walls. if collided it saves the updated direction
;and recalculate the direction according to the walls state. then sub the saved
;direction from it and update the location again
dmootmove proc dmadr:dword
        mov ebx,dmadr
        cmp [ebx + DM.animindex],-1
        je ATHOME
        cmp [ebx + DM.freeze],0
        jg FREEZED
        mov [ebx + DM.lastcoladr],ebx
        mov ecx,[ebx+DM.data]
        ;gets key’s input and updates location
        invoke keyboard, dmadr,(DmootData PTR [ecx]).spdec
        invoke updateloc, dmadr
        mov ebx,dmadr
        mov ecx,[ebx+DM.data]
        ;checks a collision with the walls
        invoke checkwalls, dmadr,(ObjData PTR [ecx]).imgw,(ObjData PTR
[ecx]).imgh
        cmp result,0
        je FREEDMOOT
        ;saves the direction
        mov ecx,dmadr
        mov ebx,[ecx + DM.drcdec.x]
        mov cos,ebx
        mov ebx,[ecx + DM.drcdec.y]
```

```
        mov sin,ebx
        ;updates the direction
        invoke handlewalls,dmadr
        ;sub the saved direction from the updated and recalculate location
        mov ebx,cos
        mov ecx,dmadr
        sub [ecx + DM.drcdec.x],ebx
        mov ebx,sin
        sub [ecx + DM.drcdec.y],ebx
        invoke updateloc, dmadr
        FREEDMOOT:

        mov ebx,dmadr
        mov ecx,[ebx+DM.data]
        invoke checkwalls,dmadr,(DmootData PTR [ecx]).homx,(DmootData PTR
[ecx]).homy
        mov eax,0
        mov al,result
        mov ebx, 3
        cdq
        div ebx
        mov ecx,dmadr
        mov [ecx+DM.transp],eax
        add [ecx+DM.hometime],eax
        mov ebx,[ecx+DM.hometime]
        mul ebx
        mov [ecx+DM.hometime],eax
        cmp eax,res.hometimelimit
        jl ATHOME
        mov [ecx + DM.transp],0;makes the Dmoot vulnerable
        ;sub [ecx + DM.score],1;takes a point out of the Dmoot score
        ;mov [ecx + DM.animindex],-1;removes the DMoot from the game
        ATHOME:
ret
        FREEZED:
        ;adds 1 to the dmoot freeze value and checks if should stay freezed
        mov ebx,dmadr
        mov ecx,1
        add [ebx + DM.freeze],ecx
        mov edx,[ebx + DM.freeze]
        cmp res.freezeTime,edx
        jg ATHOME
```

```
        mov [ebx + DM.freeze],0
        ret
dmootmove endp

;invokes dmootmove for each player in the given array
allDmootMove proc arrayadr:dword,arrayLength:dword
        AGAIN:
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayadr
        invoke dmootmove,[ebx]
        dec arrayLength
        add arrayadr,SIZEOF dword
        jmp AGAIN
        FINISHED:
ret
allDmootMove endp

;initializes the player with the init data and the CopyMem proc
;sets the player transparent and freezed
initializeDmoot proc adrdm:dword
        mov eax,adrdm
        mov [eax + DM.freeze],1
        mov [eax+DM.transp],1
        mov ebx,[eax + DM.data]
        invoke CopyMem,ebx,SIZEOF ObjInit,adrdm
ret
initializeDmoot endp

;invokes initializeDmoot for each player in the given array
initializeAllDmoot proc arrayAdr:dword,arrayLength:dword
        AGAIN:
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayAdr
        invoke initializeDmoot,[ebx]
        dec arrayLength
        add arrayAdr,SIZEOF dword
        jmp AGAIN
        FINISHED:
ret
initializeAllDmoot endp
```

```
;handle with all the events of a player: collusion with bubble and flicker
;acts correspondingly if lastcoladr is not the address of the player (it collided with a
bubble)
dmooteventhandler proc adrdm:dword,counter:dword
        mov ebx,adrdm
        ;if player is dead - FINISHED
        cmp [ebx + DM.animindex],-1
        je NOTDRAW

        ;if freezed make if flicker
        cmp [ebx + DM.freeze],0
        jg FREEZED

        ;handle the player anim
        invoke handleAnim,adrdm,counter
        ;check for a collusion
        mov ebx,adrdm
        cmp [ebx + DM.lastcoladr],ebx
        je DRAWDMOOT

        ;there was a collision
        ;check if transparanted
        cmp [ebx + DM.transp],0
        jg DRAWDMOOT
        ;if score = 0 player dies
        cmp [ebx + DM.score],0
        je DMOOTDIES
        ;decrease score and n
        mov ecx,1
        sub [ebx + DM.score],ecx
        mov ecx,[ebx + DM.score]
        dec n

        ;copy the last bubble data to the bubble the player collided with. the bubble
the player collided with will disappear
        imul ecx,n,TYPE bubbles
        add ecx,offset bubbles
        ;the bubble that disapears will be transparanted
        mov [ecx + DM.transp],1
        invoke CopyMem,ecx,TYPE bubbles,[ebx + DM.lastcoladr]
```

```
        ;reset the last collision addrres to the player's offset
        mov ebx,adrdm
        mov [ebx + DM.lastcoladr],ebx

        ;checks if no bubbles n=0
        cmp n,0
        jne REMAIN

        ;n = 1 and player dies, not draw player
        inc n
        DMOOTDIES:
        dec AlivePlayersNum
        mov ebx,adrdm
        mov [ebx + DM.animindex],-1
        jmp NOTDRAW

        REMAIN:
        ;if n>0 initialized and draw player
        invoke initializeDmoot,adrdm
        DRAWDMOOT:
        invoke drawImg,adrdm
        NOTDRAW:
ret
        FREEZED:
        ;player flickers
        ;modulu 2 of the (freeze/60) is the true/false condition to draw the player
        mov ebx,adrdm
        mov eax,[ebx+ DM.freeze]
        mov ecx,60
        cdq
        div ecx
        mov ecx,2
        cdq
        div ecx
        cmp edx,0
        je DRAWDMOOT
ret
dmooteventhandler endp

;invokes dmooteventhandler for each player in the given array
allDmootEventHandler proc arrayadr:dword, arrayLength:dword,counter:dword
        AGAIN:
```

```
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayadr
        invoke dmooteventhandler,[ebx],counter
        dec arrayLength
        add arrayadr,SIZEOF dword
        jmp AGAIN
        FINISHED:
ret
allDmootEventHandler endp;

;adds bubble and initialize the player if collide with the goal
;if the player is alive checks if collide with goal
;if collides initialize the player and increase the score and adds a bubble
dmootAddBubble proc adrdm:dword
        mov ebx,adrdm
        ;check if alive
        cmp [ebx+DM.animindex],-1
        je DONTADDBUBBLE
        invoke collcheck,adrdm,offset goal
        cmp result,1
        jne DONTADDBUBBLE
        ;initialize and increase score
        invoke initializeDmoot,adrdm
        mov ebx,adrdm
        mov ecx,1
        add [ebx+DM.score],ecx
        call addBubble
        DONTADDBUBBLE:
ret
dmootAddBubble endp

;invokes dmootAddBubble for each player in the given array
allDmootAddBubble proc arrayAdr:dword,arrayLength:dword
        AGAIN:
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayAdr
        invoke dmootAddBubble,[ebx]
        add arrayAdr,SIZEOF dword
        dec arrayLength
        jmp AGAIN
```

```
        FINISHED:
ret
allDmootAddBubble endp

;checks if there is a collision between a player and an object
;if collide it changes the player's lastcoladr to the object's offset
dmootCollCheck proc adrdm:dword,adr:dword
        invoke collcheck,adrdm,adr
        cmp result,0
        je NOcollision
        mov ebx,adrdm
        mov ecx,adr
        mov [ebx+DM.lastcoladr],ecx
        NOcollision:
ret
dmootCollCheck endp

;invokes dmootCollCheck for each player in the given array. with a given object
allDmootCollCheck proc arrayAdr:dword,arrayLength:dword,adr:dword
        AGAIN:
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayAdr
        invoke dmootCollCheck,[ebx],adr
        dec arrayLength
        add arrayAdr,SIZEOF dword
        jmp AGAIN
        FINISHED:
ret
allDmootCollCheck endp

;prints each player's score in the given array using printNum proc
AllDmootPrintScores proc posx:dword, posy:dword, arrayadr:dword,
arrayLength:dword
        AGAIN:
        cmp arrayLength,0
        je FINISHED
        mov ebx,arrayadr
        mov ebx,[ebx]
        ;check if alive
        cmp [ebx + DM.animindex],-1
        je NOTPRINT
```

```
        invoke printNum,[ebx+DM.score],posx,posy
        add posy,60
        NOTPRINT:
        dec arrayLength
        add arrayadr,SIZEOF dword
        jmp AGAIN
        FINISHED:
ret
AllDmootPrintScores endp
```

Bubble_funcs.inc

```
include Dmoot_funcs.inc
.code

;calculate the decimal energy of a bubble and adds it to energy variable
calEnergy proc adrObj:dword
	mov ebx,adrObj
	mov eax,[ebx + DM.drcdec.x]
	imul eax
	div res.deci
	mov ecx,eax
	mov eax,[ebx + DM.drcdec.y]
	imul eax
	div res.deci
	add eax,ecx
	add energy,eax
ret
calEnergy endp

;reset bubble transp, if transp = 1 meaning the bubble didn't collide, it has space to
become solid
;if transp>1 (it collided) then transp = 1. if transp = 0, it stays 0
resetbubtransp proc adrbub:dword
	mov ecx,adrbub
	mov ebx,1
	cmp [ecx + DM.transp],ebx
	jg MOV1
	mov ebx,0
	MOV1:
	mov [ecx + DM.transp],ebx
ret
resetbubtransp endp

;check the conditions for a collision between two bubbles
;result: 0 =>no collision, 1 => collision
checkconditions proc adr:dword,adr2:dword
	;basic collision check according to position
	invoke collcheck, adr,adr2
	cmp result,0
	je RETMAIN
	mov result, 0
```

```
        ;if one of the bubbles is transparent => no collusion
        invoke transphandler, adr
        invoke transphandler, adr2
        cmp result,0
        jg RETMAIN

        ;if bubbles weren't last to collide with each other
        ;prevents bubble from colliding twice in a row
        mov ebx,adr
        mov ecx,adr2
        cmp ebx,[ecx+ DM.lastcoladr]
        jne FINECOND
        cmp ecx,[ebx+ DM.lastcoladr]
        je RETMAIN
        FINECOND:
        mov [ebx + DM.lastcoladr], ecx
        mov [ecx + DM.lastcoladr], ebx

        ;if two bubbles overlap, one become transparent and ==> no collision
        invoke collisioncheck, adr,adr2,res.defensedist
        cmp result,0
        je NOTSPECIALCASE
        mov ecx,adr
        mov [ecx + DM.transp],2
        jmp RETMAIN
        NOTSPECIALCASE:
        mov result,0
ret
        RETMAIN:
        mov result,1
        ret
checkconditions endp

;calculates the cos and sin of an the collusion angle and puts it in cos and sin
variables
calangle proc adr:dword,adr2:dword
        ;calculate the squared distance between the bubbles into dist
        mov ebx,[eax+DM.pos.x]
        mov ecx,[edx+DM.pos.x]
        sub ebx,[edx+DM.pos.x]
        imul ebx,ebx
```

```
        mov ecx,[eax+DM.pos.y]
        sub ecx,[edx+DM.pos.y]
        imul ecx,ecx
        add ebx,ecx
        mov dist,ebx

        ;takes a squre root of dist using the fpu - converts it todecimal display
        fild dist
        fsqrt
        fild res.deci
        fmulp st(1), st(0)
        fistp dist

        ;cos = dx/dist (decimal)
        mov ecx,adr
        mov eax, [ecx + DM.pos.x]
        mov ecx,adr2
        sub eax, [ecx + DM.pos.x]
        mul res.deci
        invoke division, eax,dist,offset cos

        ;sin = dx/dist (decimal)
        mov ecx,adr
        mov eax,[ecx + DM.pos.y]
        mov ecx,adr2
        sub eax,[ecx + DM.pos.y]
        mul res.deci
        invoke division, eax,dist,offset sin
ret
calangle endp

;switches the velocity vectors of the bubbles that are vertical to the collision
swapVelVector proc, adr:dword,adr2:dword
        ;rotates the direction's vectors
        invoke rotate, adr
        invoke rotate, adr2

        ;switches the x values
        mov eax,adr
        mov edx,adr2
        mov ebx,[eax + DM.drcdec.x]
        mov ecx,[edx + DM.drcdec.x]
```

```
        mov [edx + DM.drcdec.x],ebx
        mov [eax + DM.drcdec.x],ecx
        neg sin

        ;rotates back
        invoke rotate, adr
        invoke rotate, adr2
ret
swapVelVector endp

;handle a collision event between two given bubbles
;checks condition - if there's a collision: calculate angle and switch the vectors
collisionHandler proc adr:dword,adr2:dword
        invoke checkconditions, adr,adr2
        cmp result,1
        je GOTOMAIN
        invoke calangle, adr,adr2
        invoke swapVelVector,adr,adr2
        GOTOMAIN:
ret
collisionHandler endp
```

Main_loops.inc

```
include Bubble_funcs.inc
.code

;adds a bubble if needed
;checks if there are players alive, if true invokes allDmootAddBubble
;if not checks if it's time to add a bubble
bubbleAdder proc counter:dword
	cmp AlivePlayersNum,0
	jng AUTOADDER
	invoke allDmootAddBubble,offset offplayers,LENGTHOF offplayers
	jmp FINISHED
	AUTOADDER:
	mov ebx,res.addbubblecount
	mov eax,counter
	cdq
	div ebx
	cmp edx,0
	jne FINISHED
	call addBubble
	FINISHED:
ret
bubbleAdder endp

;receives a bubble as an input and changes it's anim,calculate the energy,reset it's
transp,
;multiplies its direction by the energyconst,and draw it
drawInnerLoop proc adr:dword,counter:dword,energyconst:dword
	invoke handleAnim,adr,counter
	invoke calEnergy,adr
	invoke resetbubtransp,adr

	invoke muldrc,adr,energyconst

	invoke drawImg,adr
ret
drawInnerLoop endp

updateInnerLoop proc adr:dword,index:dword
	invoke handlewalls, adr
	invoke allDmootCollCheck,offset offplayers,LENGTHOF offplayers,adr
```

```
        inc index
        mov ebx,index
        cmp ebx,n
        je AFTER
        LOOP2:
        imul esi, index, TYPE bubbles
        add esi, offset bubbles ;esi is the address of the bubble j
        invoke collisionHandler,adr,esi
        inc index
        mov ebx, index
        cmp ebx,n
        jl LOOP2
        AFTER:
        invoke updateloc, adr
ret
updateInnerLoop endp

;invokes drawInnerLoop for each bubble
drawLoop proc energyconst:dword,counter:dword
        mov energy,0
        mov i,0
        loop3:
                imul edi, i, TYPE bubbles
                add edi, offset bubbles ; edi is the address of the bubble i
                invoke drawInnerLoop,edi,counter,energyconst
                inc i
                mov ebx,i
                cmp ebx,n
                jl loop3
ret
drawLoop endp
```

Main_code.inc

```
include drd.inc
includelib drd.lib
include data.inc
include Main_loops.inc
.code

load proc
        call setData
        invoke initializeAllDmoot,offset offplayers,LENGTHOF offplayers
        ;intialize the fpu
        finit
        ;load screen background
        invoke drd_init,res.wbg,res.hbg,0
        invoke drd_imageLoadFile,offset hd.bgpath,offset ram.pics.bg

        ;load num strip
        invoke drd_imageLoadFile,offset hd.numspath,offset ram.pics.nums
        invoke drd_imageSetTransparent,offset ram.pics.nums,0000000h

        invoke loadAllObjPics,offset dataStructsOffsets, LENGTHOF
dataStructsOffsets
ret
load endp

update proc counter:dword

        invoke drd_processMessages
        invoke allDmootMove,offset offplayers,LENGTHOF offplayers
        invoke bubbleAdder,counter

        mov i,0
        ;invokes updateInnerLoop for each bubble
        LOOPp:
                imul edi, i, TYPE bubbles
                add edi, offset bubbles ;edi is the address of the bubble i
                invoke updateInnerLoop,edi,i
                inc i
                mov ebx,i
                cmp ebx,n
                jl LOOPp
```

```
ret
update endp

draw proc counter:dword
	;initialze the screen
	invoke drd_pixelsClear,0
	invoke drd_imageDraw,offset ram.pics.bg,0,0

	;calculate the multiplication constant for the bubbles drc according to the
energy in the system
	mov ecx,res.deci
	mov eax,TeoreticEnergy
	mul res.energyLowBound
	div res.deci
	cmp energy,eax
	jg FINE
	mov ecx,res.energyLowConst
	jmp FINE2
	FINE:
	mov eax,TeoreticEnergy
	mul res.energyHighBound
	div res.deci
	cmp energy,eax
	jl FINE2
	mov ecx,res.energyHighConst
	FINE2:

	invoke drawLoop,ecx,counter
	call calTeoreticEnergy

	;if there is a player alive draw goal,handle players and print score
	cmp AlivePlayersNum,0
	jng SKIP
	invoke drawImg,offset goal
	invoke allDmootEventHandler,offset offplayers,LENGTHOF offplayers,counter
	invoke AllDmootPrintScores,20,150,offset offplayers,LENGTHOF offplayers
	SKIP:
	invoke drd_flip
ret
draw endp
```

Main.asm

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


main proc
	;creates a local variable which counts the number of runs
	push ebp
	mov ebp,esp
	sub esp,4
	mov DWORD PTR [ebp - 4],0
	invoke load
	loopi:
		inc DWORD PTR [ebp - 4]
		invoke update,[ebp - 4]
		invoke draw,[ebp - 4]
	jmp loopi
	mov esp,ebp
	pop ebp
ret
main endp
end main
```