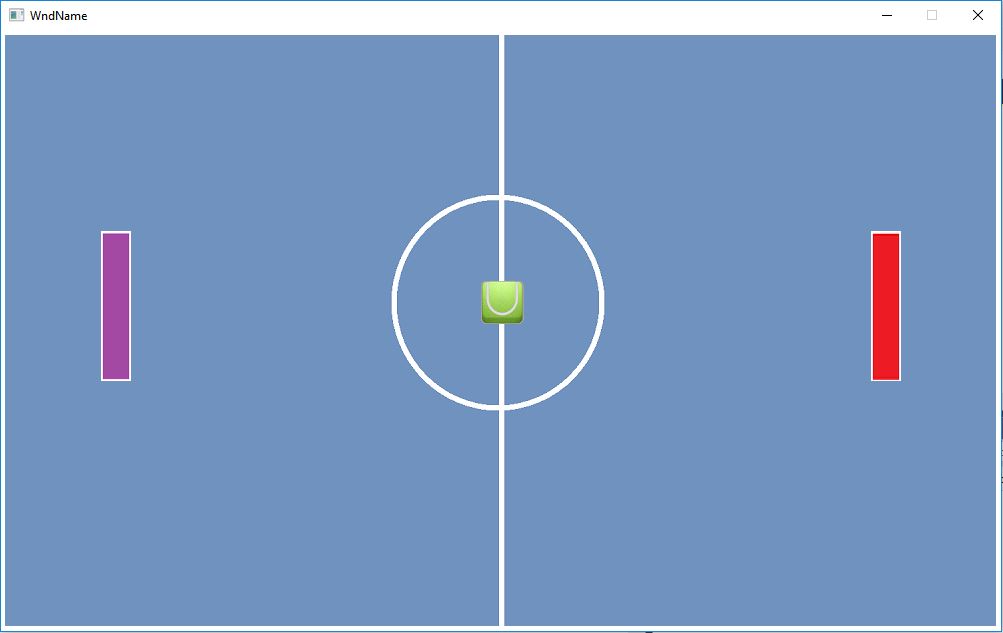
**Pong**

**פרויקט באסמבלי 32**



**מגישה: מיכל ברנופולסקי י'9**

**תעודת זהות: 322996919**

**תאריך: 25.06.2017**

**למורה: יהודה אור**

**הכפר הירוק**

**תוכן עניינים**

תוכן עניינים ............................................................................................................2

מבוא ......................................................................................................................3

קצת על אסמבלי ......................................................................................................4

פקודות בסיסיות באסמבלי .......................................................................................7

רפלקציה .................................................................................................................9

על המשחק .............................................................................................................11

תרשים הקוד ..........................................................................................................12

פונקציות ...............................................................................................................19

קוד מתועד .............................................................................................................33

ביבליוגרפיה ...........................................................................................................34

**מבוא**

קוראים לי מיכל ברנופולסקי ואני בכיתה י'9.

כחלק ממגמת הנדסת תוכנה היינו צריכים להכין השנה פרויקט בשפת התכנות אסמבלי 32, ואני החלטתי שהפרויקט הראשון שאני אכתוב אי פעם יהיה המשחק Pong.

ביצוע הפרויקט לא הייתה משימה פשוטה, וכללה הרבה למידה וגם עשייה עצמית. השקעתי בה זמן ומאמצים רבים מכיוון שזאת הפעם הראשונה שנתקלתי במשימה בסגנון זה.

הכתיבה בשפת אסמבלי הייתה אתגר בפני עצמה משום ששפת זאת הינה ישנה ומסובכת יותר משפות שאני מכירה ורגילה לכתוב בהן כמו Python או Java.

**קצת על אסמבלי**

אסמבלי (או שפת סף) היא השפה הבסיסית ביותר והקרובה ביותר לשפת מכונה הכוללות פנייה מפורשת לרכיבים במחשב, כגון אוגרים ותאי זיכרון (בדומה לשפת המכונה). בשפות תכנות אחרות כל פעולה ופעולה מתרגמות לעשרות אלפי פקודות בשפת מכונה, באסמבלי הפעולות מתרגמות לפקודה אחת בשפת מכונה.

בעבר התכנות באסמבלי היה נפוץ ודומיננטי, אך כיום משמש בעיקר לכתיבת חלקי תוכנה חשובים במערכות ההפעלה. מתכנתים משתמשים באסמבלי במקומות בהם דרושים ביצועים מהירים או שימוש יעיל במיוחד בזיכרון וכן לכותבי וירוסים או קוד המשמש לפריצה.

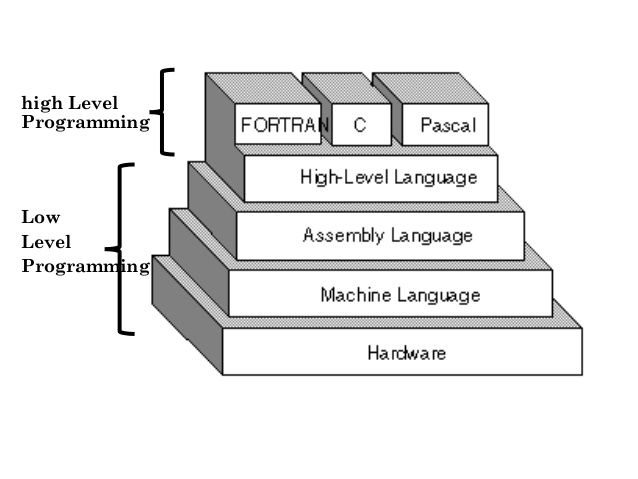
אסמבלי נחשבת לשפה מסובכת יותר משפות תוכנה חדשות יותר מכיוון שהיא אינה "ידידות למשתמש", הכתיבה בה מרובה לעומת שאר השפות. בכדי לבצע את הפעולות הפשוטות ביותר, שבשפות תוכנה עליות קיימות פעולות פשוטות והגיוניות בכדי לעשות זאת, באסמבלי דרושה כתיבה רבה. לדוגמה: הצבת סכומם של המשתנים b ו-c במשתנה a.

בשפה עילית, כמו Python או Java. מספיקה פקודה יחידה: a = b + c. באסמבלי, לעומת זאת, הפעולה תהיה ארוכה יותר:

mov eax, b - הפעולה מעבירה את הערך של B אל אוגר eax.

add eax, c - הפעולה מוסיפה את הערך של C לערך שנמצא באוגר eax.

mov a, eax - הפעולה מעבירה את הערך שנמצא באוגרeax אל המשתנה A.



**פקודות בסיסיות באסמבלי**

**פקודות בסיסיות באסמבלי**

**פקודת mov –** (move) פקודה המעתיקה את הערך של אופרנד המקור אל אופרנד היעד. השימוש העיקרי של הפקודה בקוד שלי הוא העתקת נתונים ממשתנה לאוגר ולהפך.

לדוגמה: mov eax, 50 , בהנחה ש: eax =1324 אז eax =50.

**פקודת add –** (addition) פקודת חיבור המוסיפה את הערך של אופרנד המקור אל ערך אופרנד היעד.

לדוגמה: add eax, 100, בהנחה ש: eax =43 אז eax =143.

**פקודת sub -** (subtract) פקודת חיסור המורידה את הערך של אופרנד המקור מערך אופרנד היעד.

לדוגמה: sub limitX, 500, בהנחה ש: limitX=1000 אז limitX=500.

**פקודת inc –** (increment) פקודת חיבור המוסיפה אחד לערך אופרנד.

לדוגמה: dec y, בהנחה ש: y=15 אז y=16.

**פקודת dec –** (decrement) פקודת חיסור המורידה אחד מערך אופרנד.

לדוגמה: dec x, בהנחה ש: x=15 אז x=14.

**פקודת mul –** (multiply) פקודת כפל unsigned של האוגר eax, הפקודה כופלת את הערך של האופרנד בערך של האוגר eax.

לדוגמה: mul ebx , בהנחה ש:eax=5, ebx=2 אז eax =10.

**פקודת neg –** (negative) פקודה מחליפה את הערך של אופרנד בערך המשלים שלו לפי שיטת המשלים ל-2, מחליפה את הסימן של האופרנד מחיובי לשלילי ולהפך.

לדוגמה: neg dirx3, המספר 00112 יהפוך להיות 11012.

**פקודת cmp –** (compare) פקודה המשווה בין ערכי אופרנד היעד ואופרנד המקור בעזרת חיסור הערך של אופרנד המקור מהערך של אופרנד היעד, מבלי לשנות את אופרנד היעד, אחריה בדרך כלל תהיה פקודת jmp כלשהי. השימוש העיקרי של הפקודה בקוד שלי הוא בכדי ליצור לולאות תנאי.

לדוגמה: cmp eax, limitY, ישווה בין האוגר eax והמשתנה limitY.

**פקודת jmp –** (jump) פקודת קפיצה ללא שום תנאים, הפקודה קופצת ללייבל אותה היא מקבלת. השימוש העיקרי של הפקודה בקוד שלי מיועד בכדי ליצור לולאות.

לדוגמה:jmp again , יקפוץ ללייבל again.

**פקודות תנאי –** פקודות התנאי באסמבלי מיוצגות בעזרת הקפיצות (jmp) וההשוואות (cmp), ובדרך כלל הן יבואו מיד לאחר פקודת ההשוואה – "קפוץ אם": אם דבר מסוים מתקיים בהשוואה, אז תקפוץ ללייבל הנתון.

קיימים הרבה סוגים שונים של "קפוץ אם", והם מיוצגים בעזרת האותיות:

L – lower, נמוך יותר: jl left.

G – greater, גבוהה יותר: jg right.

E – equal, שווה: je continue.

N – not, לא / שונה: jne again.

Z – zero, אפס: jz start.

**לייבל –** תווית, סוג של "מראה מקום" ובעזרתו אפשר לדעת לאן לפנות בקפיצות.

**אופרנד -** נתון כלשהו עליו פועל אופרטור.

**אופרנד המקור –** הנתון השני, הימני.

**אופרנד היעד –** הנתון הראשון, השמאלי.

**אופרטור** - ציה הפועלת על נתון (משתנה, קבוע, טיפוס וכו') כלשהו ומחזירה ערך מטיפוס מוגדר.

**רפלקציה**

בתחילת השנה כאשר כולנו הגענו לשיעור בפעם הראשונה, התחלנו ללמוד אסמבלי.

המטרה של השיעורים הייתה מאוד פשוטה – ללמוד כמה שיותר דברים כדי שאנחנו נוכל אחר כך לשחזר את מה שאנחנו יודעים, ולכתוב פרויקט בכוחות עצמנו.

בהתחלה מאוד חששתי מה יהיה, הרי זאת הפעם שראשונה האני מקבלת משימה בסגנון כזה, משימה שאני אעשה את כולה לבד. ישר התחלתי לחשוב מה אני אשעה בפרויקט, והחלטתי שאני אכתוב משחק מחשב.

לאחר התלבטויות בין משחקים רבים, אני הגעתי להחלטה שלי – את הפרויקט הזה, המשחק הראשון שאני אכתוב אי פעם יהיה המשחק Pong.

לאחר שהגעתי להחלטה שאני אעשה את המשחק Pong , חזרתי הביתה והורדתי על המחשב שלי visual studio 2013 וגם masm 32 וניגשתי לעבודה.

העבודה לא הייתה קלה בכלל ובמהלכה נתקלתי בהמוני קשיים ותקלות, אך לבסוף אחרי כמעט שנה של עבודה קשה, סיימתי את המשחק שלי.

במהלך העבודה התמודדתי עם קשיים רבים וגם למדתי איך לפתור אותם. למדתי איך נכון להוריד תוכנות למחשב וגם למדתי לכתוב באסמבלי.

לאחר שסיימתי את העבודה הרגשתי גאווה גדולה, הייתי מאוד מרוצה וגאה בעצמי שהצלחתי לכתוב משחק שלם, המשחק הראשון שלי, לגמרי לבד והלכתי ישר להוריי בשביל להראות להם על מה עבדתי במהלך כל השנה הזאת. ישבנו כולנו ביחד, אני, אמא שלי, אבא שלי ואחותי הקטנה ולכל אחד הייתה הזדמנות לשחק במשחק שלי.

אחרי שסיימנו לשחק, ניגשתי לכתיבת ספר הפרויקט.

כתיבת הספר הייתה הרבה יותר פשוטה מכתיבת המשחק, אך כתיבת המשחק היית הרבה יותר מהנה מכתיבת ספר הפרויקט.

העבודה על המשחק וגם על הספר לימדה אותי המון דברים. למדתי לחשוב מחוץ לקופסה ולהיות יצירתית, שיפרתי את יכולת הלמידה העצמית שלי והדבר הכי חשוב שלמדתי – אני מסוגלת לעשות את כל מה שאני רק רוצה.

**על המשחק**

המשחק שלי מורכב מ-3 חלקים:

* כדור.
* שחקן 1 - סגול.
* שחקן 2 - אדום.

המשחק שלי בנוי לשני שחקנים, שנצבים על המגרש ולכדור. שחקן 1 הוא הסגול ושחקן 2 הוא האדום, והם זזים בעזרת המקלדת. הכדור הוא "עצמאי", והמחשב מזיז אותו.

כל אחד מהשחקנים ניצב בצד אחר של המסך, וכל אחד מהם יכול לנוע למעלה ולמטה, אך לא לצדדים. כאשר המחבט מגיע לקצה העליון או לקצה התחתון של המסך, השחקן לא יוכל להמשיך להזיז את המחבט באותו הכיוון ויהיה חייב לשנות את כיוון התזוזה.

הכדור ניצב במקום רנדומלי בערך ה-Y שלו, וערך ה-X שלו נשאר קבוע - הכדור נשאר על הקו הלבן שחוצה את המגרש לשני חלקים שווים. כאשר הכדור פוגע בקצה העליון של המסך הוא משנה, באופן אוטומטי, את כיוון התנועה ומתחיל לנוע למטה, וכך גם להפכך כשהכדור פוגע בקצה התחתון של המסך.



המשחק מתחיל במסך הסבר שבו כתוב מהו צבעו של כל שחקן (שחקן1 סגול ושחקן2 אדום), באיזה צד של המסך הוא עומד להיות וגם על איזה מקשים כל אחד מהשחקנים צריך ללחוץ על מנת להזיז את המחבט שלו למעלה ולמטה.

המשחק מתחיל ברגע שאחד מהשחקנים לוחץ על ה"רווח".

כאשר הכדור פוגע בקצה הימני או בקצה השמאלי של המסך, נגמר המשחק. מטרת המשחק היא למנוע מהכדור להגיע אל הקצה שלך - שחקן שהכדור הצליח להגיע אל הצד שלו מפסיד, והשחקן השני מנצח.

**תרשים קוד**

הגדרת משתנים

האם "רווח" נלחץ?

כן

לא

ציור שחקן2

בדיקת שחקן1 בגבולות מסך

הגיע לקצה העליון?

הגיע לקצה התחתון?

הכדור זז

התנגשות שחקן1?

התנגשות שחקן2?

הגיע לקצה ימין?

הגיע לקצה שמאל?

בדיקת שחקן2 בגבולות מסך

שחקן1 הפסיד

שחקן 2 ניצח

שחקן1 ניצח

שחקן 2 הפסיד

תזוזה למטה

תזוזה למעלה

תזוזה למטה

תזוזה למעלה

לא

לא

לא

"S" נלחץ?

"DOWN" נלחץ?

"UP" נלחץ?

"W" נלחץ?

ציור שחקן1

לא

לא

לא

לא

כן

כן

לא

לא

כן

כן

שנה כיוון

ציור הכדור

התחלת המשחק

ציור מסך פתיחה

לא

כן

כן

כן

כן

**פונקציות**

הקוד של המשחק שלי בנוי ברובו מפונקציות שמורצות שוב ושוב בלולאת המשחק. הפעולות הראשיות במשחק הן:

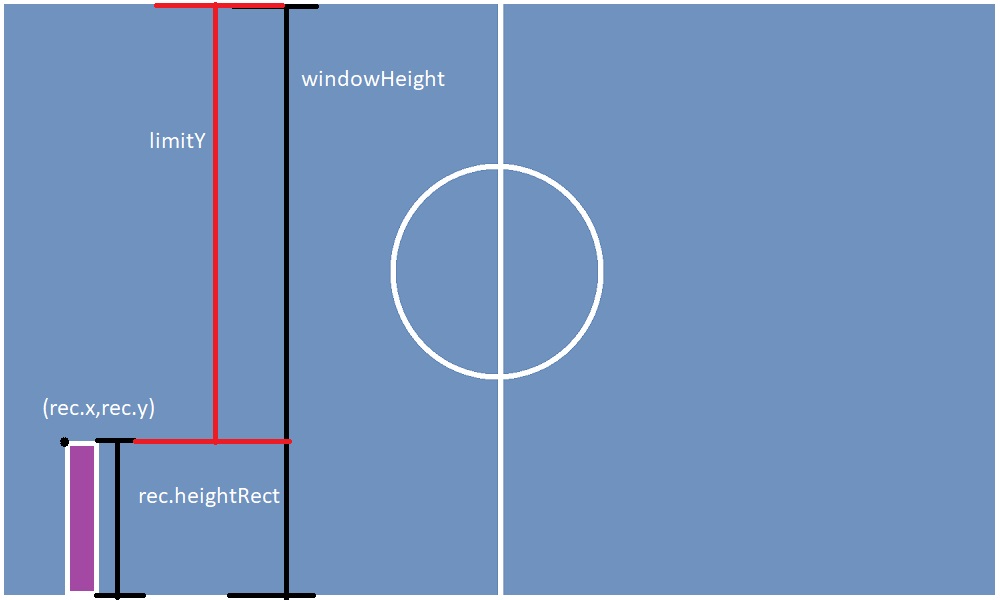
* Collison
* MovementManager1
* MovementManager2
* BallManager
* Init

**פונקציית init –** פונקציה זאת אחראית לשמור על 2 המחבטים בגבולות המסך, שלא יצאו מהגבול התחתון. הפונקציה שומרת על כך שאף אחד מהמחבטים לא יצויר מחוץ למסך, אחרת המשחק יספיק לעבוד.

פונקציית init שומרת על המחבטים בתוך גבולות המסך בדרך הבאה:

בגלל שבמהלך המשחק המחבטים זזים אך ורק לאורך ציר ה-Y, אין צורך לערוך בדיקה לשמירת המחבטים בגבולות המסך בחלק הימני והשמאל של המסך – ה-X נשאר קבוע בשני המחבטים במהלך כל המשחק. לכן צריך לעשות בדיקה רק לאורך ציר ה-Y.

קודם כל בגלל ששני המחבטים זהים בגודלם, רק הצבע משתנה מסגול לאדום בניהם, אפשר להתייחס רק ל-struct של אחד מהם (אני בחרתי לכתוב את הפעולה בהתייחסות רק ל-rec ולא ל-rec2). הפונקציה מחסירה את האורך של המחבט מאורך התמונה בגלל שצריך להתייחס למחבט לפי הנקודה העליונה בצד שמאל למעלה, מורידה עוד 5 פיקסלים (תורם לדיוק הגבול ולמסך יש גבולות לבנים שאני לא רוצה שהמחבט יגיע אליהם), ואז מעבירה את הערך שהתקבל כגבול של התנועה לאורך ציר Y למשתנה limitY.



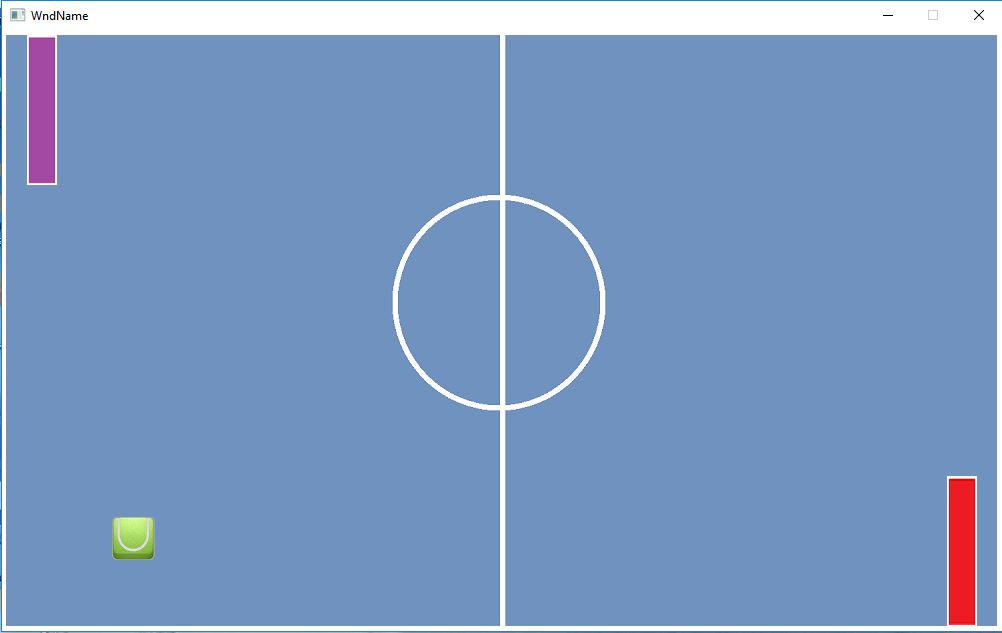
**פונקציות MovementManager1 ו-MovementManager2 –** פונקציות אלו אחראיות לזימון התזוזה של המחבטים עם רווח הזמן הנכון בין תזוזה לתזוזה על מנת שהשחקנים יספיקו להבין מה מתרחש, בכל אחד מהמחבטים (MovementManager1 - סגול, MovementManager2 - אדום).

הפונקציה מגדילה בכל פעם את המשתנה turn, שמשמש כסוג של מהירות, באחד ואז משווה אותו למספר שאני קבעתי מראש כדי "לכוון" את רווח הזמן (ככל שמשתנה זה יותר מושווה למספר יותר גדול התזוזה יותר איטית).

כשהמשתנה turn מגיעה לגודל הרצוי הפונקציה נכנסת לתוך לולאה שבה היא עושה את התזוזה של כל אחד מהמחבטים בהתאם. בלולאה אני בודקת איזה מקש נלחץ, והאם בכלל נלחץ אחד ממקשי התנועה, ואם כן אני משנה את כיוון התנועה בהתאם ללחיצת המקש ואז קוראת לפונקציהMove1 ולפונקציה Move2 (MovementManager1 קוראת ל- Move1 , ו- MovementManager2 קוראת ל- Move2).

**פונקציות Move1 ו-Move2 –** פונקציות אלו אחראיות לביצוע התזוזה בכל אחד מהמחבטים בהתאם, רק לאחר שפונקציות הMovementManager מזמנות אותן.

למיקום ה-Y של כל אחד מן המחבטים מתווסף המשתנה diry (כיוון התנועה, או 1+ או 1-) וכן מזיזה את המחבט פיקסל אחד בהתאם למקש שנלחץ. לאחר מכן כל אחת מהפעולות בודקת שהמחבט לא חרג מגבולות המסך בעזרת המשתנה borderY (לכל אחד מהמחבטים קיים borderY משל עצמו). כאשר המחבט מגיע ל-Y=4 (אני בחרתי את הגבול הזה כי יש מסגרת למסך) אז הפונקציה קופצת לעצירת התנועה לפי borderY , ומשם המחבט לא ימשיך לזוז עד שהכיוון לא יוחלף.



**פונקציות BallX ו- BallY -** פונקציות אלו אחראיות לביצוע התזוזה בכל אחד מהצירים, ציר ה-X וציר ה-Y, של הכדור רק לאחר שפונקציות ה-BallManager מזמנת אותן.

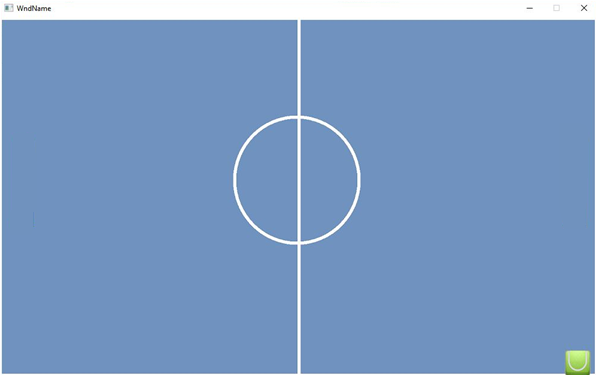
בגלל שהתזוזה של הכדור מופעלת על ידי המחשב, ולא על ידי אחד מהשחקנים, לפני תחילת התזוזה חובה לבדוק קודם כל לאן נע הכדור (בציר ה-Y למעלה או למטה, ובציר ה-X ימינה או שמאלה). לאחר הבדיקה לפי הכיוונים (diry ו-dirx), כל אחת מהפונקציות קופצת בהתאם ללולאה אותו היא צריכה לבצע.

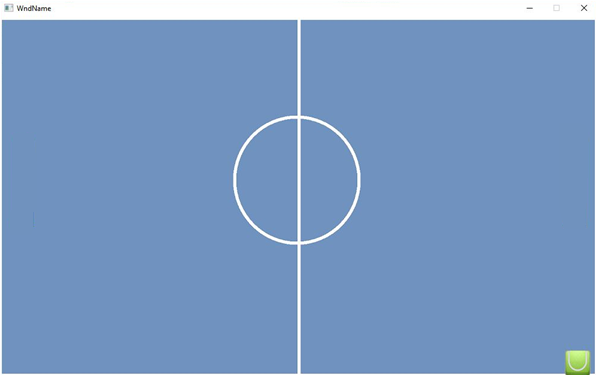
הפונקציות מגדילות את המשתנה turn ב-1 בכדי להזיז את הכדור פיקסל אחד לכיוון הרצוי ואז קופצת ללולאת continue (שומרת את הכדור במסך בעזרת בדיקת מיקום הכדור עם אורך/רוחב המסר פחות אורך/רוחב הכדור בהתאם) ולאחר מכן משנה את כיוון התנועה בהתאם לתוצאת הלולאה.

**פונקצייתBallManager –** פונקציה זו אחראית לזימון התזוזות של הכדור עם רווח הזמן הנכון בין תזוזה לתזוזה על מנת שהשחקנים יספיקו להבין מה מתרחש, בכל אחד מהצירים, ציר ה-X וציר ה-Y.

הפונקציה מגדילה בכל פעם את המשתנה turnBall, שמשמש כסוג של מהירות, באחד ואז משווה אותו למספר שאני קבעתי מראש כדי "לכוון" את רווח הזמן (ככל שמשתנה זה יותר מושווה למספר יותר גדול התזוזה יותר איטית).

כשהמשתנה turnBall מגיעה לגודל הרצוי הפונקציה נכנסת לתוך לולאה שבה היא עושה את התזוזה של הכדור בכל אחד מהצירים בהתאם - קוראת לפונקציהBallX ולפונקציה BallY ומאפסת את turnBall.



**פונקציות EndGame1 ו- EndGame –** פונקציות אלו אחראיות לציור התמונות על המסך כאשר נגמר המשחק. יש 2 אפשרויות ציור – שחקן1 מנצח ושחקן2 מפסיד (EndGame1), ו-שחקן1 מפסיד ושחקן2 מנצח (EndGame).

כל אחת מהפונקציות מציירת 2 תמונות: אחת של מנצח ואחרת של מפסיד, במיקומי X ו-Y שקבועים מראש לכל אחד מהמקרים.

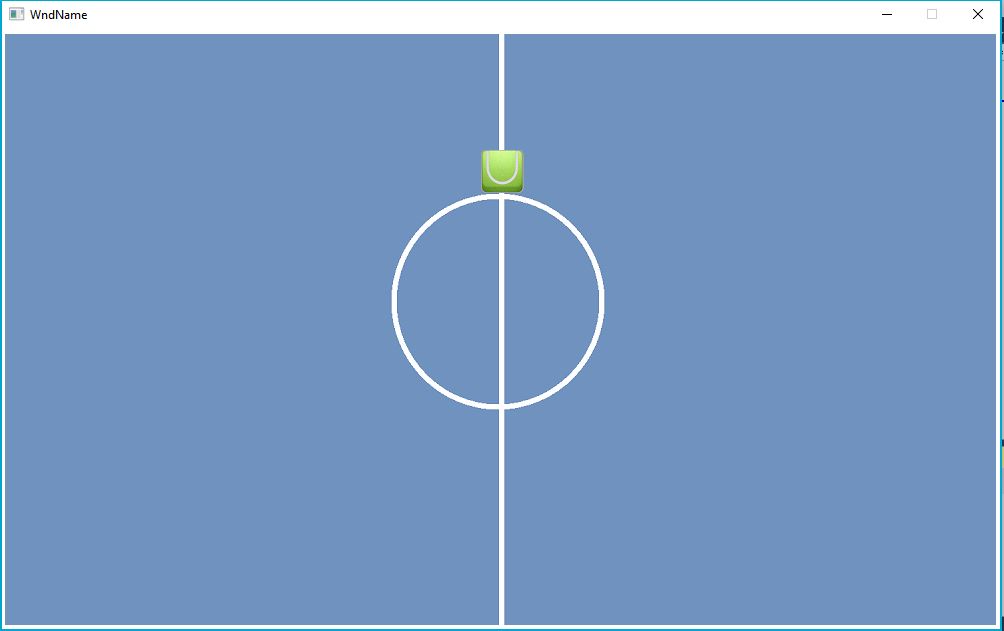
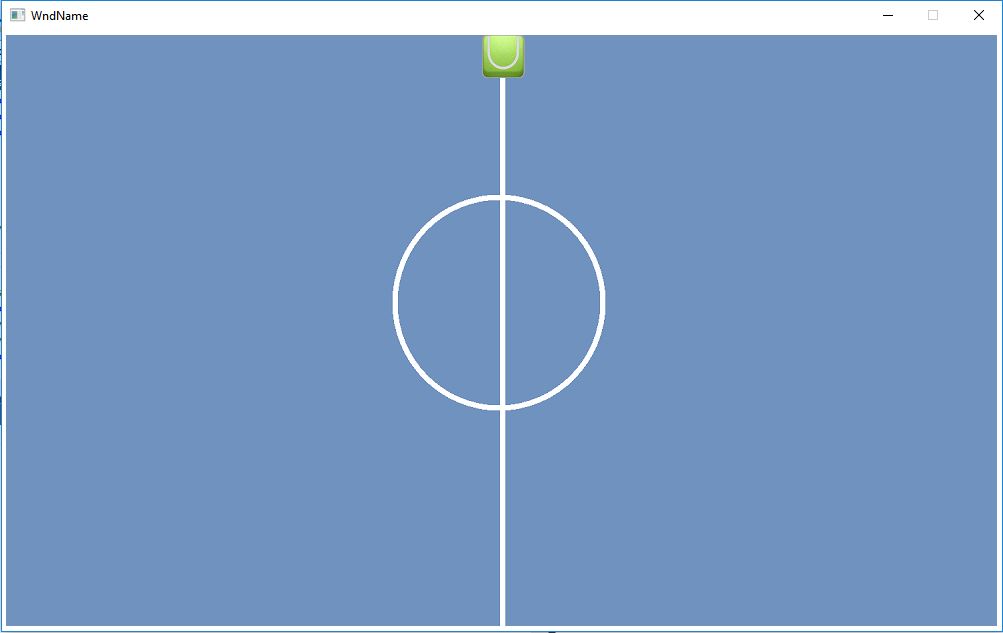
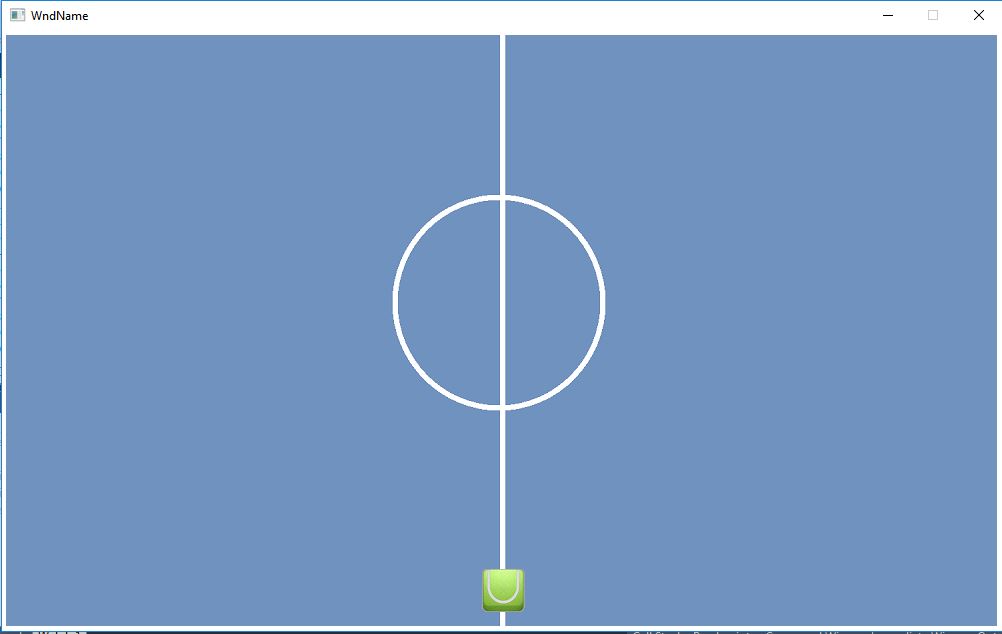
מקרה EndGame1: שחקן1 ניצח, שחקן2 הפסיד.

מקרה EndGame: שחקן1 הפסיד, שחקן2 ניצח.

**פונקציית Random –** פונקציה זו אחראית למתן ערך Y רנדומלי בטווח שאני קבעתי מראש, לפי מסגרת המסך ובהתייחסות לאורך הכדור, מראש של מ-1 עד ל-557.

הערך הרנדומלי שמתקבל בפונקציה נשמר במשתנה random\_num ולאחר מכן הוא מועבר אל tnn.y, הערך הרנדומלי מועבר אל ערך ה-Y ההתחלתי של הכדור לפני תחילת המשחק (ולפני הציור), וכך בכל משחק הכדור יהיה מצויר במקום שונה בציר ה-Y.

מספר דוגמאות למקומות בהם הכדור יכול להיות מצויר:



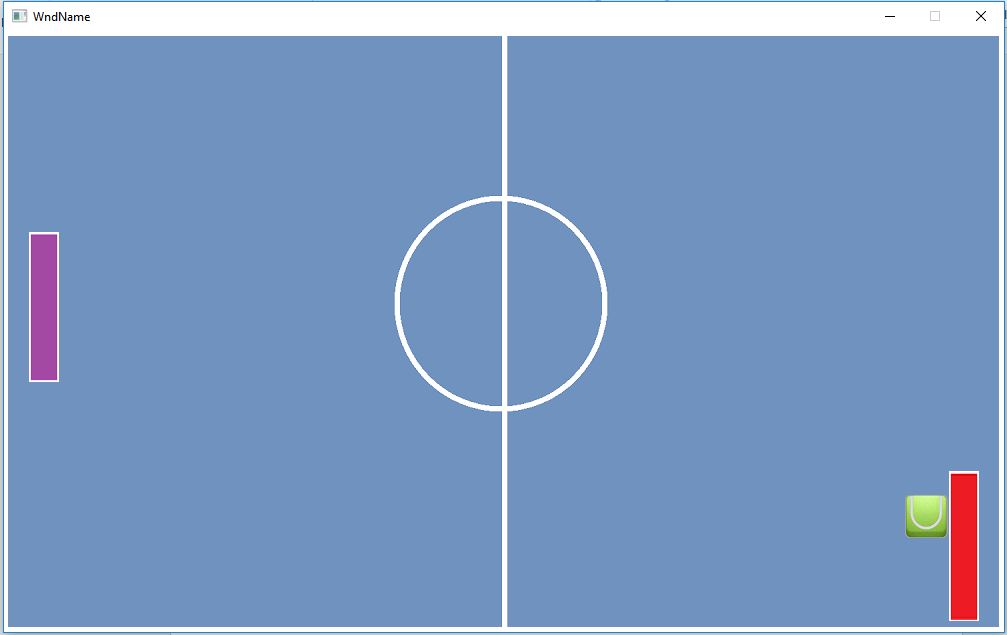
לפי התמונות ניתן לראות, שבכל אחת מהפעמים הכדור מצויר במיקום Y שונה, אך ערך ה-X של הכדור **תמיד** נשאר קבוע – הכדור תמיד מצויר באמצע המסך, על הקו הלבן, בדיוק כמו שאני הגדרתי אותו מראש.

**פונקציית Collision –** פונקציה זו היא הפונקציה העיקרית (רוב המשחק מתרחש בעזרתה) והחשובה ביותר במשחק. היא אחראית ליצירת ההתנגשות בין הכדור לבין שני המחבטים, בעזרת הפרויקט נהיה משחק – הפונקציה הזאת גורמת ליצירת מטרה ברורה מאוד ומאפשרת לשחק בפרויקט.

תחילה הפונקציה בודקת לאיזה כיוון נע הכדור (בשביל לדעת עם איזה מחבט תהיה ההתנגשות) ואז היא קופצת ללולאה המתאימה (שמאל או ימין).

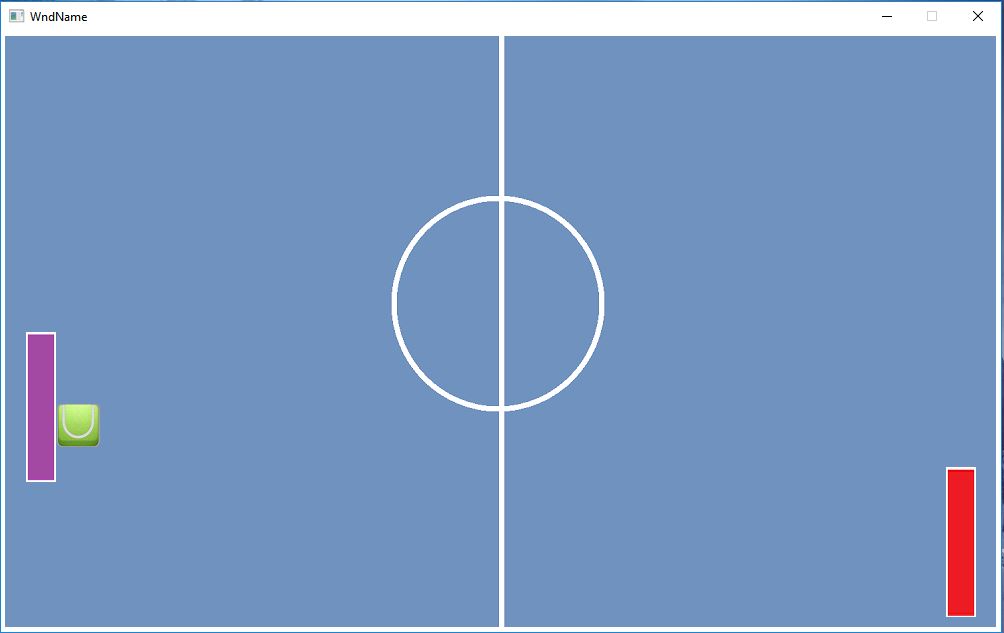
צורת הכתיבה בבדיקת הטווחים היא: קודם אני אבדוק מה לא נכון (ואעשה לו jmp exit) ולאחר מכן, אם זה הגיע עד לנקודה הזאת בדוק, זאת אומרת שיש התנגשות.

**לולאת ימין –** קודם כל הפונקציה בודקת את ערכי ה-X של הכדור ושל שחקן2, בבדיקה זו **צריך** להתחשב ברוחב הכדור, (הרי אם הערכים שונים אין טען להמשיך ולבדוק אם קיימת התנגשות בין השניים), ואם ערכי ה-X זהים (או מאוד קרובים, כי לפעמים יכולה להיות סטייה של פיקסל אחד או שניים) הפונקציה עוברת לבדיקת ערכי ה-Y של הכדור וגם של שחקן2 .

קודם כל אני משווה החלק העליון של הכדור אל החלק התחתון של שחקן2, בעזרת הוספת אורך המחבט, ואם החלק העליון של הכדור גדול (נמוך יותר) מהחלק התחתון של המחבט אני יודעת שאין התנגשות, והמשחק ממשיך כרגיל.

אם החלק העליון של הכדור קטן (גבוה יותר) מהחלק התחתון של המחבט אני ממשיכה הלאה בלולאה ובודקת את החלק התחתון של הכדור, בעזרת הוספת אורך הכדור, עם החלק העליון של המחבט. אם החלק התחתון של הכדור קטן (גבוה יותר) מהחלק העליון של המחבט אני יודעת שאין התנגשות והמשחק ממשיך כרגיל.

ואם לא, אז יש התנגשות (יאיי), ואז אני משנה את כיוון תנועת הכדור ועוברת ללולאת שמאל.



**לולאת שמאל -** קודם כל הפונקציה בודקת את ערכי ה-X של הכדור ושל שחקן12, בבדיקה זו **לא צריך** להתחשב ברוחב הכדור, (הרי אם הערכים שונים אין טען להמשיך ולבדוק אם קיימת התנגשות בין השניים), ואם ערכי ה-X זהים (או מאוד קרובים, כי לפעמים יכולה להיות סטייה של פיקסל אחד או שניים) הפונקציה עוברת לבדיקת ערכי ה-Y של הכדור וגם של שחקן1.

לאחר בדיקת ערכי ה-X מתקיימת בדיקה זהה לבדיקת ההתנגשות בלולאת ימין.

**קוד מתועד**

.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

.data

STime SYSTEMTIME{}; at: proc Random

;Variables for the Random:

;The rundom number - the starting Y value of the ball.

random\_num DWORD 0

;The minimal Y value.

mini DWORD 1

;The maximal Y value.

maxi DWORD 557

;The size of the window.

windowWidth DWORD 1000

windowHeight DWORD 600

;The limits of the window.

limitX DWORD windowWidth

limitY DWORD windowHeight

;Stopping position on the Y direction.

borderY DWORD 0

borderY2 DWORD 0

;The delay between the movements.

turn DWORD 0

turnBall DWORD 0

;The settings of the background.

pong BYTE "Pong.bmp",0

Back struct

info Img<>

;backgroung starting positon.

x DWORD 0

y DWORD 0

;the size of the background.

widthBack DWORD 1000

heightBack DWORD 600

Back ends

bg Back<>

;The settings of the ball.

tennis BYTE "Tennis.bmp",0

Ball struct

info Img<>

;The ball's starting position.

x DWORD 480

y DWORD 250

;The starting direction of the ball.

dirx DWORD 1

diry DWORD 1

;The size of the ball.

widthBall DWORD 43

heightBall DWORD 43

Ball ends

tnn Ball<>

;The setting of the player1 - purple racket.

rect BYTE "Player.bmp",0

Player1 struct

info Img<>

;Player1's starting directions.

dirx DWORD 1

diry DWORD 1

;Player1's starting position.

x DWORD 25

y DWORD 200

;The size of Player1's racket.

widthRect DWORD 30

heightRect DWORD 150

Player1 ends

rec Player1<>

;The settings of player2 - red racket.

rect2 BYTE "Player2.bmp",0

Player2 struct

info Img<>

;Player2's starting directions.

dirx DWORD 1

diry DWORD 1

;Player2's starting position.

x DWORD 945

y DWORD 200

;The size of Player2's racket.

widthRect DWORD 30

heightRect DWORD 150

Player2 ends

rec2 Player2<>

;The settings of Win.

winner BYTE "Win.bmp",0

WinImg struct

info Img<>

;Win starting position.

x DWORD 500

x1 DWORD 0

y DWORD 0

;The size of win image.

widthRect DWORD 500

heightRect DWORD 600

WinImg ends

winn WinImg<>

;The settings of Lose.

lose BYTE "Lose.bmp",0

LoseImg struct

info Img<>

;Lose starting position.

x1 DWORD 500

x DWORD 0

y DWORD 0

;The size of lose image.

widthRect DWORD 500

heightRect DWORD 600

LoseImg ends

looser WinImg<>

;The settings of the starting window.

start BYTE "PongWelcome.bmp",0

StartingWindow struct

info Img<>

;PongWelcome starting position.

x DWORD 0

y DWORD 0

;The size of PongWelcome image.

widthRect DWORD 1000

heightRect DWORD 600

StartingWindow ends

stt StartingWindow<>

.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

;Random Y value for the ball.

Random PROC min:DWORD, max:DWORD

; "A pseudo-random number generator (PRNG) is a program that takes a starting number (called a seed),

; and performs mathematical operations on it to transform it into some other number that appears to be

; unrelated to the seed." - learncpp.com

;

; random = (seed \* some big number) % (max - min + 1) + min

;

; by ohad gitzelter 0546677598

pusha

; calculates max - min + 1

mov eax, max

sub eax, min

mov max, eax

inc max

; gets a always changing seed

invoke GetSystemTime ,addr STime

; moves to ax the current clock miliseconds and moves the bx the current clock seconds,

; the two values that changes most frequently

mov ax, STime.wMilliseconds

mov bx, STime.wSecond

; multiplies eax by the current miliseconds and seconds (ax and bx)

; (seed \* some big number)

mul ax

mul bx

; reset edx to prevent integer overflow (you shouldnt care about why)

xor edx, edx

; divides the value in eax by the max value (reminder will be in edx)

; (seed \* some big number) % (max - min + 1)

div max

; add min to the final result

; + min

add edx, min

; move the random number in edx to your desired variable

mov random\_num, edx

popa

ret

Random ENDP

;This is the movement of the ball along the X direction.

BallX PROC

pusha

;Checks what direction the ball is haeding to (Left or Right).

cmp tnn.dirx, 0

jg right

jl left

;Moves the ball 1 pixel to the left.

left:

mov eax, tnn.x

dec eax

mov tnn.x, eax

jmp continue

;Moves the ball 1 pixel to the right.

right:

mov eax, tnn.x

inc eax

mov tnn.x, eax

;Keeps the ball in the window.

continue:

;Subtracts the ball's width from the window width.

mov eax, windowWidth

sub eax, tnn.widthBall

;Checks to which direction the ball needs to face.

cmp tnn.x, eax

popa

jge goLeft

cmp tnn.x, 5

jl goRight

jmp exit

;Changes the balls direction.

goLeft:

mov tnn.dirx, -1

jmp exit

;Changes the balls direction.

goRight:

mov tnn.dirx, 1

jmp exit

exit:

ret

BallX ENDP

;This is the movement of the ball along the Y direction.

BallY PROC

pusha

;Checks what direction the ball is haeding to (Up or Down).

cmp tnn.diry, 0

jg up

jl down

;Moves the ball 1 pixel down.

down:

mov eax, tnn.y

dec eax

mov tnn.y, eax

jmp continue

;Moves the ball 1 pixel up.

up:

mov eax, tnn.y

inc eax

mov tnn.y, eax

;Keeps the ball in the window.

continue:

;Subtracts the ball's width from the window width.

mov eax, windowHeight

sub eax, tnn.heightBall

;Checks to which direction the ball needs to face.

cmp tnn.y, eax

popa

jge goUp

cmp tnn.y, 5

jl goDown

jmp exit

;Changes the balls direction.

goUp:

mov tnn.diry, -1

jmp exit

;Changes the balls direction.

goDown:

mov tnn.diry, 1

jmp exit

exit:

ret

BallY ENDP

BallManager PROC

pusha

add turnBall,1

popa

;turn is like the speed. The bigger the number that I compare to gets, the slower the ball moves.

cmp turnBall, 4

je doTurn

jmp exit

;Does the movement.

doTurn:

mov turnBall, 0

invoke BallX

invoke BallY

exit:

ret

BallManager ENDP

;This is the movement of player1 along the Y direction.

MoveY1 PROC

pusha

;Moves player1 1 pixle.

mov eax, rec.y

add eax, rec.diry

mov rec.y, eax

;Checks if player1 reached the bottom limit of the window.

mov eax, limitY

cmp rec.y, eax

mov borderY, eax

jg stop

;Jumps to stop when player2 is on Y = 4.

cmp rec.y, 4

mov borderY, 4

jl stop

jmp exit

;Stops player1 in Y = borderY.

stop:

mov eax, borderY

mov rec.y, eax

exit:

popa

ret

MoveY1 ENDP

;Activates MoveY1.

MovementManager1 PROC

pusha

;Increases turn by one.

mov eax, turn

inc eax

mov turn, eax

;The delay between the movements.

cmp turn, 5

je doTurn

popa

jmp exit

doTurn:

popa

mov turn, 0

;Checks if the "W" key pressed.

invoke GetAsyncKeyState, VK\_W

cmp eax, 0

;Changes the direction and goes to the movement.

mov rec.diry, -1

jne MoveY1

;Checks if the "S" key pressed.

invoke GetAsyncKeyState, VK\_S

cmp eax, 0

;Changes the direction and goes to the movement.

mov rec.diry, 1

jne MoveY1

exit:

ret

MovementManager1 ENDP

;This is the movement of player2 along the Y direction.

MoveY2 PROC

pusha

;Moves player2 1 pixle.

mov eax, rec2.y

add eax, rec2.diry

mov rec2.y, eax

;Checks if player2 reached the bottom limit of the window.

mov eax, limitY

cmp rec2.y, eax

mov borderY2, eax

jg stop

;Jumps to stop when player2 is on Y = 4.

cmp rec2.y, 4

mov borderY2, 4

jl stop

jmp exit

;Stops player2 in Y = borderY2.

stop:

mov eax, borderY2

mov rec2.y, eax

exit:

popa

ret

MoveY2 ENDP

;Activates MoveY2.

MovementManager2 PROC

pusha

;Increases turn by one.

mov eax, turn

inc eax

mov turn, eax

;The delay between the movements.

cmp turn, 5

je doTurn

popa

jmp exit

doTurn:

popa

mov turn, 0

;Checks if the "W" key pressed.

invoke GetAsyncKeyState, VK\_UP

cmp eax, 0

;Changes the direction and goes to the movement.

mov rec2.diry, -1

jne MoveY2

;Checks if the "S" key pressed.

invoke GetAsyncKeyState, VK\_DOWN

cmp eax, 0

;Changes the direction and goes to the movement.

mov rec2.diry, 1

jne MoveY2

exit:

ret

MovementManager2 ENDP

;Keeps the rectangle inside the window.

init PROC

pusha

;Moves the height of the window to: limitY by the register.

mov eax, windowHeight

mov limitY, eax

;Subtract the rectangle's height from the window height.

mov eax, limitY

sub eax, rec.heightRect

sub eax, 5

mov limitY, eax

popa

ret

init ENDP

Collision PROC

pusha

;Checks which direction the ball moves.

mov eax, tnn.dirx

cmp eax, 0

jg right

jl left

;Ckecks for a collision with player2

right:

;Adds the width of the ball.

mov eax, tnn.x

add eax, tnn.widthBall

;Checks if the ball and player2 are at the same X location.

cmp eax, rec2.x

jge overXr

jmp exit

;Checks if the ball in the Y range of player2.

overXr:

;Adds the height of playet2.

mov eax, rec2.y

add eax, rec2.heightRect

;Compares the top of the ball with the bottom of player2.

cmp tnn.y ,eax

jg exit

;Adds the height of the ball.

mov eax, tnn.y

add eax, tnn.heightBall

;Compares the bottom of the ball with the top of player2.

cmp eax, rec2.y

jl exit

;Changes the direction of the ball.

mov tnn.dirx, -1

jmp left

;Ckecks for a collision with player2

left:

;Adds the width of the player1.

mov eax, rec.x

add eax, rec.widthRect

;Checks if the ball and player1 are at the same X location.

cmp tnn.x, eax

jle overXl

jmp exit

;Checks if the ball in the Y range of player1.

overXl:

;Adds the heigth of player1.

mov eax, rec.y

add eax, rec.heightRect

;Compares the top of the ball with the bottom of player1.

cmp tnn.y, eax

jg exit

;Adds the height of the ball.

mov eax, tnn.y

add eax, tnn.heightBall

;Compares the bottom of the ball with the top of player1.

cmp eax, rec.y

jl exit

;Changes the direction of the ball.

mov tnn.dirx, 1

exit:

popa

ret

Collision ENDP

EndGame PROC

;Draws the image at the end of the game.

;Player1 lost and Player 2 won.

invoke drd\_imageDraw, offset looser.info, looser.x1, looser.y

invoke drd\_imageDraw, offset winn.info, winn.x, winn.y

ret

EndGame ENDP

EndGame1 PROC

;Draws the image at the end of the game.

;Player1 won and Player 2 lost.

invoke drd\_imageDraw, offset looser.info, looser.x, looser.y

invoke drd\_imageDraw, offset winn.info, winn.x1, winn.y

ret

EndGame1 ENDP

main PROC

;Draws a window according to given size.

invoke drd\_init, bg.widthBack, bg.heightBack , 0

;Loads all the images.

invoke drd\_imageLoadFile, offset tennis, offset tnn.info

invoke drd\_imageLoadFile, offset rect, offset rec.info

invoke drd\_imageLoadFile, offset rect2, offset rec2.info

invoke drd\_imageLoadFile, offset pong, offset bg.info

invoke drd\_imageLoadFile, offset lose, offset looser.info

invoke drd\_imageLoadFile, offset winner, offset winn.info

invoke drd\_imageLoadFile, offset start, offset stt.info

invoke drd\_imageSetTransparent, offset tnn.info, 0a0a0a0h

invoke Random, mini, maxi

invoke drd\_imageDraw, offset stt.info, stt.x, stt.y

invoke drd\_processMessages

invoke drd\_flip

starti:

;Moves the value of random\_num to the stating Y position of the ball.

mov eax, random\_num

mov tnn.y, eax

;Checks if "Space" key pressed.

invoke GetAsyncKeyState, VK\_SPACE

cmp eax, 0

jne again

jmp starti

again:

;Checks if the ball reched the left edge.

cmp tnn.x,6

jle endGame

;Checks if the ball reched the right edge.

mov eax, windowWidth

sub eax, tnn.widthBall

cmp eax, tnn.x

jne continue

invoke EndGame1

jmp over

endGame:

invoke EndGame

jmp over

continue:

;Draws the images on the screen according to the given X and Y position.

invoke drd\_imageDraw, offset bg.info, bg.x, bg.y

invoke drd\_imageDraw, offset tnn.info, tnn.x, tnn.y

invoke drd\_imageDraw, offset rec.info, rec.x, rec.y

invoke drd\_imageDraw, offset rec2.info, rec2.x, rec2.y

invoke Collision

invoke init

invoke MovementManager1

invoke MovementManager2

invoke BallManager

over:

invoke drd\_processMessages

invoke drd\_flip

jmp again

ret

main ENDP

end main

**ביבליוגרפיה**

פעולות בסיסיות באסמבלי. *Atmel, AVR Assembler Instructions*:

<http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html>

שפת סף, באנגלית ובעברית (עודכן לאחרונה ב-10.03.2017). *ויקיפדיה*, האנציקלופדיה החופשית:

<https://en.wikipedia.org/wiki/Assembly_language>

גונן, ב' (מהדורה שנייה 2015). *ארגון המחשב ושפת סף*, ספר גבהים:

<http://data.cyber.org.il/assembly/gvahim_assembly_book.pdf>

YodaSatla (2017). GoogleDrive:

<https://drive.google.com/drive/>